

Виктор Филинков
Курсы по программированию для мам, пап и их детей

вступление Виктора к курсу — [здесь](#)

(текст обновляется и дополняется; последнее обновление - 24.04.2021, до Занятия 80)

Оглавление

Занятие 0.1. О том, что я задумал	5
Занятие 0.2. Как заниматься	6
МОДУЛЬ 1	8
Занятие 1. Значёчки [ещё в пути]	8
Занятие 2.1. Множество	8
Занятие 2.2. Интерпретаторы	11
Занятие 2.3. Python	11
Занятие 3.1. Задания	12
Занятие 3.2. KBD	12
Занятие 4.1. Отношения	13
Занятие 4.2. Типы объекта	14
Занятие 4.3. Переменные	14
Занятие 5. Задания	15
Занятие 6.1. Подпрограммы	16
Занятие 6.2. Степени	16
Занятие 6.3. Типизация	17
Занятие 8.1. Функция в смысле языка программирования	17
Занятие 8.2. Настройки Python'a	18
Занятие 8.3. Задания	19
Занятие 10.1. Классы	19
Занятие 10.2. Ввод и вывод	21
Занятие 10.3 Создание классов	22
Занятие 15.1. Деление	22
Занятие 15.2. Присвоение и конструктор	23
Занятие 15.3. Паттерны программирования	24
Занятие 16.1. Задания	26
Занятие 16.2. Задания	26

МОДУЛЬ 2	26
Занятие 20.1. Свёртка и упорядоченные пары	26
Занятие 20.2. Прелести и радости. Абстрактное “Мяу”	28
Занятие 20.3. Нейминг	30
Занятие 21. Истина и Ложь	31
Занятие 22. Задания	33
Занятие 23.1. Английский язык	34
Занятие 23.2. Как заниматься #2	35
Занятие 26. Техническое задание на проект ‘хо’ (версия 0.1)	35
Занятие 30.1. Ветвление	37
Занятие 30.2. Системы счисления	39
Занятие 30.3. Задания	40
Занятие 31.1. Задания	40
Занятие 31.2. Задания	40
Занятие 35.2. Схемы алгоритмов	40
Занятие 35.3. Задания	41
Занятие 36.1. Задания	41
Занятие 36.2. Задания	42
Занятие 36.3. Чётность-нечётность	42
МОДУЛЬ 3	42
Занятие 40.1. Шаблоны	42
Занятие 40.2. О наследовании	44
Занятие 40.3. Книга “Графы и их применение”	44
Занятие 43.1. Импликация	45
Занятие 43.2. Таблица истинности функций	45
Занятие 43.3. Массив данных	47
Занятие 46.1. Эквивалентность	47
Занятие 46.2. Циклы	49
Занятие 46.3. Массивы в цикле	51
Занятие 49.1. Выполнение логических операций	52

Занятие 49.2. О связи между операциями над множествами и логическими операциями.	53
Занятие 49.3. Бесконечный цикл	54
Занятие 51.1. Операции сравнения	54
Занятие 51.2. Двоичный код	55
Занятие 52.1. Управляющие символы	56
Занятие 52.2. Приведение типов	57
Занятие 55.1. Интерфейсы и абстрактные классы	58
Занятие 55.2. Операционная система и разрядность CPU	62
Занятие 56. Пишем 'хо'	62
Занятие 58. XOPlayer	65
Занятие 59.1. Foreach	67
Занятие 59.2. Foreach на неупорядоченном множестве	68
Занятие 60. XOView	69
Занятие 62. XOField #1	70
Занятие 63. RAM	73
Занятие 64. XOField#2	73
Занятие 65. Цикл For (для)	74
Занятие 66. XOController	76
Занятие 67. Регистры; шина	79
Занятие 68. хо.ру	79
Занятие 70. XOField #3	80
Занятие 71.1. Адресация RAM	81
Занятие 71.2. Итераторы	81
Занятие 72. XOField #4	82
Занятие 74. XOField #5	83
Занятие 75. Тестовое. Карточки #1	87
Занятие 76. Тестовое. Карточки #2	88
Занятие 77. Стек и куча. Задание	88
Занятие 80. Указатель	90
Занятие 81. ТЗ хо v0.2	92

Занятие 83. Системы сборки	92
Занятие 84. Числа с плавающей запятой	92
Занятие 86. Статические и динамические библиотеки	92
Занятие 96. Синтаксис C++	92
Занятие 98. Явное и неявное приведение типов. nullptr	93
Занятие 99. Умные указатели и прочий ужас	95
Занятие 100. Константные методы	99
Занятие 101. Часть 1. Утечка памяти	101
Занятие 101. Часть 2. Системные вызовы	101
Занятие 102. Тернарный оператор	101
Занятие 104. Константные переменные, аргументы	102
Занятие 105. Передача аргумента по значению	107
Занятие 106. Ключевое слово using. Анонимное пространство имён	107
Занятие 107.1. хо v0.3	113
Занятие 107.2. Кортеж	114
Занятие 107.3. Называем числа	116
Занятие 110. Передача аргумента по ссылке	123
Занятие 115. Контейнер и его методы. Размер	125
Занятие 119. std::string и конкатенация	127
Занятие 120. Инкремент и декремент	127
Занятие 121. strict.sizeof и числа	127
Занятие 122. Контейнер и его методы	128
Занятие 125. Контейнер и его методы. Доступ	128
Занятие 130. Контейнер и его методы. Вставка	130
Занятие 131. Выравнивание и размер структур	131
Занятие 132. Контейнер и его методы. Удаление	131
Занятие 135. Священный список.	133
Занятие 136. tc. Тестирование контейнеров	133
Занятие 137. Священный список	135
Занятие 138. Улучшение свяженного списка	136
Занятие 139. Итераторы и священные списки	137

Занятие 140. Контейнер и его методы. Доступ #2	138
Занятие 145. Мой компьютер	139
Занятие 148. Использование памяти	139
Занятие 150. Покрытие функций uint-тестами	139
Занятие 151. Linked List и его хвост	142
Занятие 155. tc и юнит-тестирование	142
Занятие 157. Тестируем хо	148
Занятие 158. Тестируем хо (2)	149
Занятие 161. Double Linked List	151
Занятие 160. Контейнер и его методы. Конструкторы	152
Занятие 163. Контейнер и его методы. Копирование	152
Занятие 162. Контейнер и его методы. Поиск.	152
Занятие 164. Код возврата программы и ехес	154
Занятие 166. Тестируем хо(3)	154
Занятие 167. Тестируем хо(4)	154
Занятие 170. ok-lock v01	156
Занятие 178. Нех и ok-lock v0.1.2	157
Занятие 180. Версионирование	157
Занятие 181. ok-lock TUI	158

Занятие 0.1. О том, что я задумал

Давай я сначала расскажу, что я задумал:

0.1

Программисты бывают разные:

- Кодер (coder) — могут перевести чёткую, формальную мысль на какой-то ЯП [язык программирования]. Никаких специфических знаний не требуется — только знание ЯП.
- Прогеры (programer) — умеют реализовывать идеи, модули.
- Разрабы (developer) — разрабатывают программные продукты целиком, т.е. могут в архитектуру.
- Инженеры (engineer) — некст левел разработчика ПО, могут делать всё.

Это всё очень условно, конечно, я вот с трудом проведу грань между девелопером и инженером, а девелоперов часто именуют и прогерами и кодерами. Ориентироваться будем на Engineer. Двигаться сразу в нескольких направлениях: математика, архитектура ПО, ЯПы, английский и ещё много сопутствующего.

Я буду предполагать, что у тебя какой-то Debian-based дистрибутив Linux, например Ubuntu, а если нет — то ты сможешь сам разобраться чё делать (ещё же есть Linux4Win!)

Что касается матеши — я буду касаться только базы, имеющей отношение к делу и только теории. Практических заданий, наверное, не будет.

По архитектуре: решать здесь будет практика. Сейчас я вижу 5-7 учебных проектов, при помощи которых мы попробуем наступить на разные грабли и рассмотрим разные аспекты разработки ПО. Двигаться будем от простого к сложному, по началу архитектуру буду описывать я — это носит рекомендательный характер, в виде примера, а последнее слово — за тобой.

О ЯП. ЯП — не важны. Уметь программировать — нечто большее, чем знать ЯП. Я буду использовать псевдокод, а ты должен понимать его абстрактно и применять свои знания при помощи любого ЯП. Мы пойдём вершками по следующим ЯП, для начала:

*Python потому, что очень просто и ничего там учить не надо, всё банально, легко гуглится, терпимое ООП и ФП.

*C++ Учить мы его не будем, ибо никто его не знает. Нужен он нам для примера статической типизации и близости к железу.

*JavaScript - ну, это WEB, все дела.

*Haskell.

Соответственно задача звучит так:

Научиться правильно выбирать ЯП для решения задачи и уметь выполнять задачу на любом ЯП.

Английский: де-факто стандарт в разработке ПО. Нас будет интересовать технический уровень: чтение документации, уметь правильно давать имена объектам и т.п. Я попрошу тебя установить словарик, что-то удобное, вроде GoldenDict и англ-рус, рус-англ словари.

Также поговорим и об IDE, по контролю версий, о пакетировании и о прочем. Про первые два проекта расскажу сразу.

Первый — “Крестики-нолики”, или ‘хо’, второй — часы, назовём ok-lock для потехи (just for fun). Для этих простеньких проектов ЯП буду выбирать я. Подумаем позже, какие проекты сгодятся для портфолио — их мы будем публиковать на GitHub. Коснёмся всего: терминала, GUI, WEB, backend.

Самого важного, к сожалению, я не смогу тебе дать. Самое важное — это мотивация. Всё будет не получаться, казаться неправильным, недостаточно хорошим — это ок, это так и задумано. Мы работаем на будущее. Сквозь тернии к звёздам.

Главный навык, который ты приобретёшь — способность быстро учиться. Также мы разовьём дисциплину, выявим твои специфические качества — как положительные, так и отрицательные (мы научимся с ними жить). Избавим тебя от страха нового: на все задачи ты будешь говорить: "А я уже делал что-то подобное!"

Занятие 0.2. Как заниматься

Как заниматься:

Я буду просить присылать мне то-то и то-то. Сразу оговорюсь: никаких контрольных, я вас не проверяю; слать мне надо для того, чтобы я понимал, куда двигаться дальше, т.к. ампула препода по разработке для меня ново, а ещё я три года в тюрьме — и ни черта сам не помню. А что-то и не знал никогда.

Что я от вас ожидаю: что вы будете уделять этому времени не реже 3-х раз в неделю, причём 4 раза по 45 мин лучше 3-х раз по часу.

Не рассчитывайте, что все освоите за месяц. Перед вами — длинный путь, он несёт в себе сложности и массу интересного. Готовы ли вы его пройти?

Заниматься лучше вечером. Просто читать по 10 раз не надо, эффективнее и быстрее *вспоминать*. После материала часто даны вопросы для самоконтроля. Можно поглядеть на них перед чтением, а сразу после чтения — попытаться ответить. Ещё хорошо пытаться вспомнить утром то, что делали вчера вечером.

► Помни: вечером учёба, утром вспоминёба! (Албанская мудрость) ◀

Не стоит делать больше одного Занятия за раз. Также, если вы не сильно торопитесь, не стоит делать больше 5 Занятий в неделю. При этом нет ничего плохого в том, чтобы идти вперёд, если всё понятно. Результатом успешных занятий является навык, то есть способность решения практических задач. Если у вас получается выполнить задания — значит всё хорошо.

Если что-то не даётся, то можно вернуться к этому позднее (но обязательно вернуться!), а сейчас пойти вперёд. Всё обязательно получится! Это касается как понимания теории, так и выполнения заданий.

Сконцентрируйтесь на материале. Не отвлекайтесь. Можно слушать спокойную фоновую музыку, если она не ест ваше внимание. Неплохие рекомендации вы найдёте также в [моей статье](#) о работе из дому в период королева-вируса.

Если что-то не понятно или не приходит нужная идея — отвлекитесь. Расфокусируйте внимание. Послушайте любимую песню, поглядите любимый клип. Можно выбрать рандомный клип из [плейлиста Ви Фи](#).

Занятия нумерованы. Большой номер означает, что для понимания может потребоваться информация из занятий с меньшим номером.

Для начала полностью просмотрите материал, который я даю, прочти все вопросы и задания. Не начинай сразу делать практические задания, но если хочется что-то попробовать, "поиграться" — не возбраняется. Вопросы будут делиться на те, которые ты должен будешь выяснить — т.е. найти инфу самостоятельно и рассказать мне (шоб я тоже знал, хех!), и тестовые на которые ты должен уметь отвечать — это нужно вспоминать, для того, чтобы запомнить. Предлагаю такой формат: сначала ты пытаешься на них ответить сразу после освоения теории, а потом ещё раз на следующем, ээ, в следующий раз как сядешь — на следующий день, например. Ошибки не важны, важен факт вспоминания и формулирования. Ответ: хочешь вслух проговаривай, хочешь — письменно. Если письменно — буду рад, если ты пришлешь — чтобы я понимал, насколько ты понял. Вообще я буду рад всем-всем черновикам от тебя, даже с каракулями и ошибками.

Соруpaste. Смотреть чужой код можно, копировать — нельзя. Т.е. ты должен будешь сам посимвольно вводить то, что подглядишь. Крайне желательно ещё понимать, как работает скопированное. Я буду рад услышать вопросы.

Так, к следующим вопросам я попрошу тебя возвращаться после каждого, э, занятия:

1. Что ты уже знал, а что оказалось в новинку? Может, что-то стало более понятным?
2. Что осталось не до конца ясным?
3. Что было сложно, а что — совсем легко?

Всё, теперь читай теорию, она отдельно.

Дополнение:

Больше двух дней перерыва между учебными занятиями — перебор. Регулярность, уверенное, постепенное движение к цели — залог вашей победы. Но если вы решите отдохнуть недельку — никто не умрет.

МОДУЛЬ 1

И начнем мы этот курс с небольшого количества математики: вспомним, что такое функция, множество, деление и степень. Настроим текстовый редактор для более приятной работы с кодом. Познакомимся с базовыми понятиями программирования: переменная, подпрограмма, класс. Начнем представлять себе программу в общих чертах. Напишем ваши первые строки на Python.

У нас будет не так много практических задач по программированию на первых занятиях. Я извиняюсь за это. Возможным решением проблемы может стать какой-нибудь курс по Python для начинающих, с большим количеством несложных заданий, желательно “прямо в браузере” с автоматической проверкой выполнения. (Если кто найдет такой, дайте ссылку.)

Занятие 1. Значёчки

[Будет добавлено позже, т.к. потерялось пока шло почтой - прим.ред.]

Занятие 2.1. Множество

- ▣ Множество (Set [set]) - это способ думать о неупорядоченных группах объектов (Object ['obdʒɪkt]).

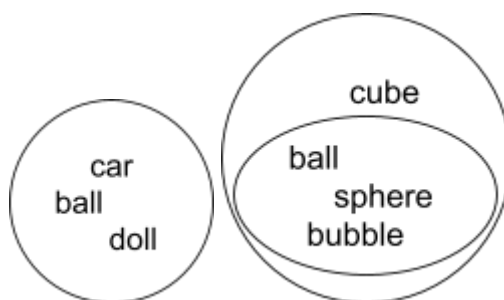
`{car, ball, doll}`

Мы будем обозначать Множество фигурными скобками (braces ['breisiz]), а ▣элементы (Element ['ɛlɪm(ə)nt]) множества будем разделять запятой (comma ['kɒmə]).

Разумеется, множество — тоже объект и может быть элементом Множества:

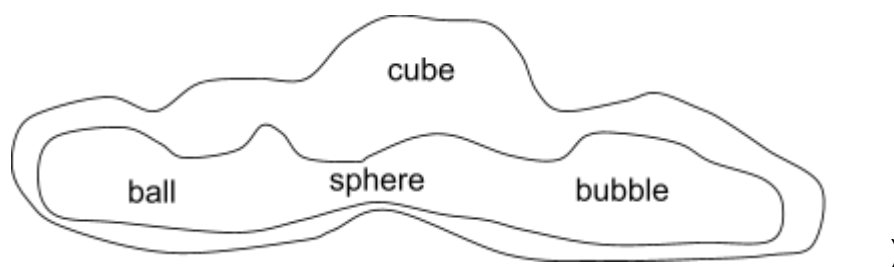
`{cube, {ball, sphere, bubble}}`

Мы можем также обозначать Множества кругами:



Иногда это удобно.

(Если круги совсем неровные, то это называется диаграммой Эйлера-Венна:



Множество без элементов называется \square пустым множеством (empty set ['empti set]) и обозначается \emptyset (перечеркнутый круг, не путать с нулём 0).

С Множествами можно проводить некоторые \square операции (Operation [opə'reiʃ(ə)n]):

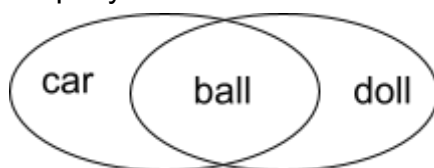
1. \square Объединение (union ['ju:njən] of sets) \cup

$$\{\text{car}, \text{ball}\} \cup \{\text{doll}\} = \{\text{car}, \text{ball}, \text{doll}\}$$

Если элемент Множества входит в оба, то в итоге не повторяется:

$$\{\text{car}, \text{ball}\} \cup \{\text{ball}, \text{doll}\} = \{\text{car}, \text{ball}, \text{doll}\}$$

Это хорошо видно на рисунке:



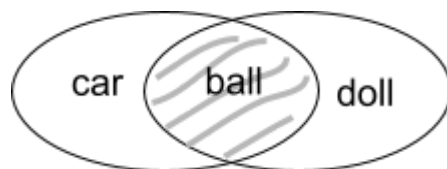
2. \square Пересечение (intersection [ɪntə'sekʃ(ə)n] of sets) \cap



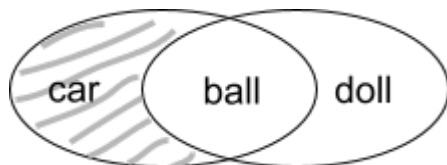
$$\{\text{car}, \text{ball}\} \cap \{\text{doll}\} = \emptyset$$

$$\{\text{car}, \text{ball}\} \cap \{\text{ball}, \text{doll}\} = \{\text{ball}\}$$

То есть пересечение — это общие для Множеств элементы



3. □Разность (difference of sets) \setminus (читается “без”)
 $\{car, ball\} \setminus \{doll\} = \{car, ball\}$
 $\{car, ball\} \setminus \{ball, doll\} = \{car\}$



4. □Дополнение (addition [ə'dɪʃ(ə)n] of set) \neg
 □Универсум или □Универсальное Множество (Universal Set [ju:nɪ'vɜ:s(ə)l set]) это множество всех элементов, которые мы договариваемся рассматривать в контексте какой-то конкретной задачи или приложения, обозначается буквой \mathcal{U} .

Дополнением Множества называется \mathcal{U} без этого Мн.:

$$\neg\{car\} = \mathcal{U} \setminus \{car\}$$

Дополнение \emptyset это \mathcal{U} (и наоборот): $\neg\mathcal{U} = \emptyset$, $\neg\emptyset = \mathcal{U}$

Говорят, что объект □принадлежит (Belong [bɪ'lɒŋ]) Множеству, если он является его элементом:

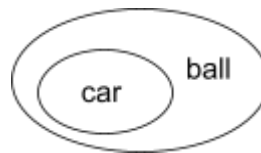
$$\underbrace{car}_{\text{car принадлежит}} \in \{car, ball\}$$

Не принадлежит — \notin

$$doll \notin \{car, ball\}$$

Говорят, что Множество □включено как подмножество (included as a subset) в другое Множество, если все его элементы являются элементами другого Множества:

$$\{\text{car}\} \subseteq \{\text{car}, \text{ball}\}$$



Очевидно, что $\{\text{doll}\} \subseteq \{\text{doll}\}$.

□ Включение называется строгим (strict inclusion, proper inclusion) (\subset), если множества, связанные им, неравны: $\{\text{car}\} \subset \{\text{car}, \text{ball}\}$, но $\{\text{car}, \text{ball}\} \not\subset \{\text{car}, \text{ball}\}$, так как они □ равны (equal ['i:kw(ə)]).

Мы можем давать названия Множествам (name [neim], имя):

```
cars = {Subaru, Honda, Lada}
```

Тогда: $Lada \in Cars$

```
{Subaru} ⊆ Cars
```

```
Cars \ {Honda} = {Subaru, Lada}
```

```
{Honda} ∪ {Subaru} ∪ {Lada} = Cars
```

```
Cars ∩ {Lada} = {Lada}
```



Занятие 2.2. Интерпретаторы

Интерпретаторы

Научи меня:

- Что такое интерпретируемый язык программирования (ЯП)?
- Что такое интерпретатор?
- Что такое исходный код?

Занятие 2.3. Python

Первым языком программирования будет Python. Когда я говорю Py, я имею в виду Py3. В моё время это ещё Python 3.5, что там сейчас я хз. Про Python 2.7 — забудь, это legacy.

Python 3

1. Установите продвинутый текстовый редактор с подсветкой синтаксиса, например Kate, Geany или VSCode. Для убунтоводов первые два в репах (“sudo apt install geany” в консоли), Kate — стандартный текстовый редактор в KDE (то есть уже предустановлен в Кубунту), а VS Code скачивается в виде .deb пакета с сайта. Виндузятикам рекомендую сразу Notepad++.
2. Установите интерпретатор Python 3. У линуксоидов он уже установлен.
“python3” в консоли чтоб запустить в режиме командной строки (Ctrl+C чтобы закрыть (выйти) консольное приложение)

“python3 hello.py” для запуска файла hello.py из домашней директории (гуглите “cd команды в терминале”, “домашняя директория”).

Виндузятники — гуглите, страдайте.

- Осильте написать и запустить “HelloWorld” на Python.

Занятие 3.1. Задания

Задания:

- $A = \{RED, BLUE\}$, $B = \{GREEN\}$, $C = \{BLUE, BLACK\}$.

- $D = A \cup B = ?$
- $D \cap C = ?$
- $B \setminus C = ?$
- $A \cap B = ?$
- $(D \cup C) \setminus B = ?$

- Пусть $\mathcal{U} = \{0, 1, 2\}$

тогда:

- $\neg\{0\} = ?$
- $\neg\emptyset = ?$
- $\neg\{2, 0\} \cap \{1, 0\} = ?$

- Почитайте про импорт символов в Python

Занятие 3.2. KBD

В этом занятии мы будем учиться слепому печатанию.

Как говорит Википедия “Слепой метод набора — методика набора текста «вслепую», то есть не глядя на клавиши пишущей машинки или клавиши клавиатуры, используя все (или большинство) пальцы рук. Ранее был известен как американский слепой десятипальцевый метод. Существует уже более 130 лет. В XIX веке слепым методом печати на пишущих машинках обучали машинисток и секретарей. Это позволило сузить сферу использования стенографии, увеличить производительность труда секретарей”.

А теперь этому будем учиться мы, чтобы быстро печатать коды, статьи, текста, рассказы, письма.

Привожу вам ссылку на клавиатурный тренажёр Klavaro: [Klavaro Touch Typing Tutor](#)

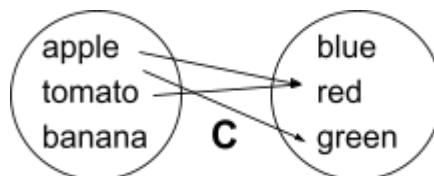
На их сайте можно найти открытый исходный код, бинарные пакеты или портативные версии для установки программы (как на винду, так и на убунту). Сама программа несёт в себе уроки разного уровня сложности (начиная с обучения, где вас научат правильно держать руки над клавиатурой, заканчивая скоростной печатью, онлайн-соревнованиями пользователей). Программа интернациональна (несёт в себя обучение на разных раскладках и языках), вы сможете отслеживать график вашего прогресса.

Как и в любом обучении главное — регулярность, уверенное, постепенное движение к цели.

Занятие 4.1. Отношения

▫Отношение (Relation) — это некоторое соответствие между двумя Множествами. Будем изображать стрелками.

Эти три стрелки образуют отношение, назовём его C:



$$C = \{apple \rightarrow red, tomato \rightarrow red, apple \rightarrow green\}$$

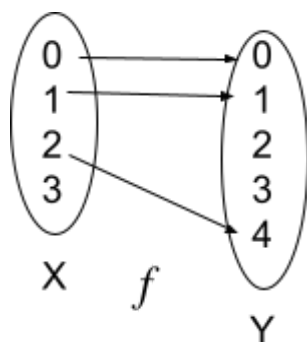
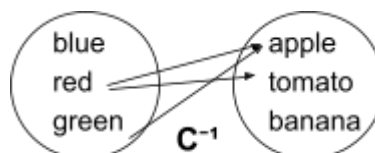
Множество всех объектов слева от \rightarrow называется \square областью определения (domain [də(ʊ)'meɪn]) (D): $D(C) = \{apple, tomato\}$. Читается “C определено на {apple, tomato}”.

А вот $banana \notin D(C)$, и значит C не определено на banana. \square Область значений (value area ['vælju: 'eəɪə]) (R) отношения — все объекты справа от \rightarrow :

$$R(C) = \{red, green\}. \text{ Значения C лежат на } \{red, green\}.$$

▫Функция (Function ['fʌŋ(k)(ə)n]) — отношение, в котором ни одна стрелка не выходит из одного и того же объекта дважды. C — не функция.

▫Обратное отношение (inverse relation) получается, если поменять D и R местами (обозначается $^{-1}$): $D(C^{-1}) = R(C)$, $R(C^{-1}) = D(C)$



Рассмотрим функцию f и два Множества X и Y ($X = \{1, 2, 3\}$, $Y = X \cup \{4\}$)

Говорят, что функция f \square отображает (to map) Множество X в Y.

Будем записывать это так:

$$f: X \rightarrow Y \quad (\square\text{“эф это функция из икс в игрек”})$$

Слова функция и отображение - синонимичны.

Заметим:

$$D(f) \subseteq X$$

$$R(f) \subseteq Y$$

Занятие 4.2. Типы объекта

Тип (type [taɪp]) объекта — это название Множества, к которому принадлежит объект.

Например: `Color = {red, green, blue}`. Значит, что тип объекта `blue` — это `Color`, то есть `blue ∈ Color`. Будем писать:

$$\text{type}(\text{red}) = \text{type}(\text{blue}) = \text{Color}$$

Несколько важных Множеств:

- Натуральные числа \mathbb{N} . Это числа 1, 2, 3 и т.д. Также мы будем считать 0 (ноль) натуральным числом. $\mathbb{N} = \{0, 1, 2, 3, 4, \dots\}$
- Множество целых чисел \mathbb{Z} . Это все положительные числа, ноль и все отрицательные числа (=положительные со знаком минус):

$$\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, 3, \dots\}$$

Очевидно $\mathbb{N} \subseteq \mathbb{Z}$.

Соответствующие им типы будем называть `unsigned integer` [ʌn'saɪnd 'ɪntɪdʒə] или просто `uint` [ju:'ɪnt / u:'ɪnt] и `signed integer` или просто `int`. “Беззнаковое” / “знаковое” целое; речь о знаке “-”.

▣ Перечисления (Enumerations [ɪnju:mə'reɪʃ(ə)n])

Перечисление — это множество объектов одного типа, каждый элемент которого отличен от всех других.

enum Color:
RED
GREEN
BLUE

Важно здесь то, что если $x \in \text{Color}$ и $x = \text{BLUE}$, то $x \neq \text{RED}$ и $x \neq \text{GREEN}$. Перечисление является типом и если тип некоторой переменной `c` объявлен как `Color`, то $c \in \text{Color}$, то есть `c` это либо `Color.RED`, либо `Color.GREEN`, либо `Color.BLUE` и четвертого не дано.

Научи меня:

- ▣ Как реализовать перечисление в Python? Есть ли для этого встроенные синтаксические средства в языке или же это неполноценный заторможенный рак среди ЯП?

Занятие 4.3. Переменные

- ▣ Переменные (variable ['veriəbl]) в языках программирования.

Переменные в языке программирования это просто имена. Имена, называющие объекты. Переменные всегда имеют тип, то есть объект, который они называют, принадлежит какому-то множеству. В динамически типизируемых языках тип переменной не указывается явно (но он есть), а просто пишется что вот это — переменная:

```
var x = -1
```

в каких-то языках программирования не нужно и ключевое слово var:

```
x = -1
```

В обоих случаях тип переменной x — это `int`. (`integer` - целое число)

Переменные могут менять своё значение — для этого им нужно просто присвоить новое:

```
x = 5
```

```
x = 100500
```

Задание:

Научиться создавать переменные в Python и менять их значения.

Занятие 5. Задания

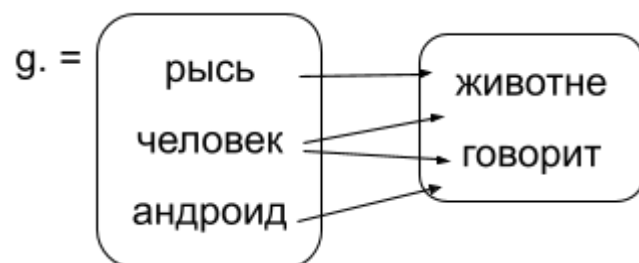
А. Изобразите кругами отношения:

1. {дочь → кошка}
2. {мама → дочь, мама → сын, папа → дочка, папа → сын}
3. Обратное к 2.
4. {мышь → кошъ, кошъ → мышъ}

Какие из них обладают свойством функциональности?

В. Изобразите в виде множеств:

$$f. = \boxed{0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4}$$



Выпишите $D(f)$, $R(f)$. Верно ли, что g и g^{-1} — функциональные отношения?

Занятие 6.1. Подпрограммы

Программы состоят из подпрограмм. Под подпрограммой мы будем понимать набор действий. Подпрограммы бывают двух видов: процедуры и функции. Различаются они тем, что функции возвращают значение, то есть некоторый результат своей работы. Процедуры значений не возвращают.

Во многих современных языках и один, и второй вид подпрограмм именуют функциями. В случае, когда процедуру называют функцией, говорят, что такая функция возвращает void (англ: “пустота”) или просто “функция не возвращает значение”. Мы переймём эту традицию и также будем называть все подпрограммы функциями. Но не стоит забывать, что существует разница между функциями в математическом смысле (функция как набор “стрелочек”, отношение между множествами) и функциями в смысле языка программирования (набор действий).

Вопросы для самоконтроля:

- + Какие бывают подпрограммы?
- + Что составляется из подпрограмм?
- + Что такое функция в смысле языка программирования?

Занятие 6.2. Степени

▫ Степень числа (Power of number [ˈpaʊə] [ˈnʌmbə]) это умножение числа сам на себя. Запись n^m означает, что число n перемножено m раз:

$$\underbrace{n \cdot n \cdot \dots \cdot n}_{m \text{ раз}} \text{ “эн в степени эм”}$$

Любое число в степени 0 это единица:
 $n^0 = 666^0 = 5^0 = 0^0 = 1$

▫ Степени десятки (decimal-system exponent [ˈdesɪm(ə)l] [ˈsɪstəm] [ɪkˈspəʊnənt]).

10^n — это 1 и n нулей. Например:

$$10^3 = 1000, 10^6 = 1000000, 10^2 = 100, 10^0 = 1, 10^1 = 10$$

Степени двойки. Особо важными в IT являются степени двойки.

Необходимо их выучить

$2^0 = 1$	$2^3 = 8$	$2^6 = 64$	$2^9 = 512$
$2^1 = 2$	$2^4 = 16$	$2^7 = 128$	$2^{10} = 1024$
$2^2 = 4$	$2^5 = 32$	$2^8 = 256$	$2^{11} = 2048$

Это просто. У нас на руках 10 пальчиков. Загибаем один, это 2^1 , то есть 2. Загибаем второй: $2 \cdot 2 = 4$, третий: $4 \cdot 2 = 8$, так далее. на одной руке $2^5 = 32$, на двух $2^{10} = 1024$.

Задания:

1. Запомнить степени двойки до 2^{10} .
2. Самостоятельно досчитать до 2^{16} (на листочке, разумеется)
3. Загуглить (закалькуляторить) 2^{32} , 2^{64} .

P.S. Есть хороший консольный калькулятор — `calc`. В Ubuntu пакет с ним называется `arcalc` или `asalc`.

P.S. 2^{64} можно посчитать в Python.

Занятие 6.3. Типизация

Научи меня:

- Что такое динамическая и статическая типизация? Приведите примеры языка программирования для одной и второй.
- Почему динамическая типизация это плохо для сложных проектов?

Занятие 8.1. Функция в смысле языка программирования

Объявить функцию означает заявить о её существовании. Будем на нашем псевдоязыке обозначать $\bar{\square}$ объявление функции (function declaration [$'f\eta(k)(\emptyset)n\ dekl\emptyset'rei'(\emptyset)n]$) так:

$$\underbrace{\text{bar:}}_{\square\text{имя}} \underbrace{X}_{\square\text{типа аргумента}} \rightarrow \underbrace{Y}_{\square\text{типа возвращаемого значения}}$$

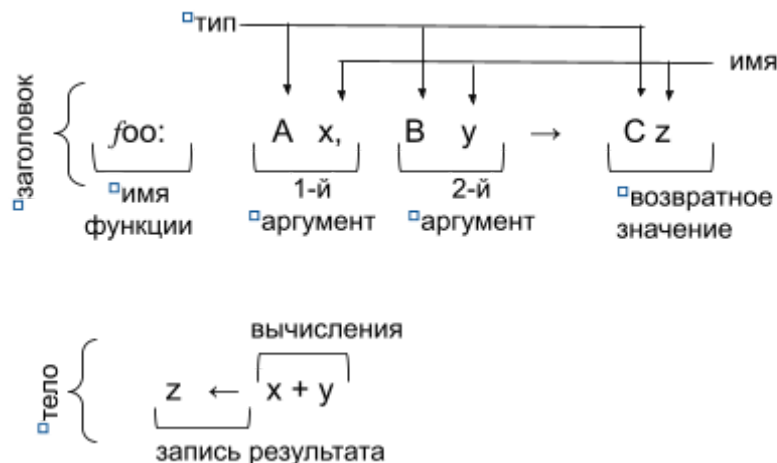
Оно означает, что существует функция `bar` с одним аргументом типа X , возвращающая значение типа Y .

Аналогичная математическая функция могла бы быть записана так:

$$\text{bar}(x), x \in X, R(\text{bar}) \subseteq Y$$

Объявление ничего не говорит о том, как работает функция.

- Определение функции (function definition [$'f\eta(k)(\emptyset)n\ defi'ni'(\emptyset)n]$):



Заголовок идентичен объявлению, но в нём также указаны имена переменных (аргументов и возвращаемого значения) для того, чтобы иметь к ним доступ в теле функции.

В нашем случае тело `foo` состоит из двух последовательных действий:

1. Вычисление суммы переменных `x` и `y`
2. Запись результата в `z`

Мы будем называть запись результата работы функции в переменную — возвращаемое значение — возвратом значения (return value [ri'tɜ:n 'væljʊ:]). Итогом работы любой функции является возврат значения (return).

Аналогичную математическую функцию можно записать так:

$$foo(x, y) = x + y, \quad x \in A, \quad y \in B, \quad R(foo) \subseteq C$$

Выполнение функции мы будем называть вызовом функции (function call [ˈfʌŋ(k)(ə)n kɔ:l]) и обозначать “()”:

`foo(1,1)` это два. Как видно, запись аналогична математической.

В отличие от математических функций, функции в языках программирования могут принимать 0 аргументов. То есть не принимать аргументов. Примером такой функции может быть функция, возвращающая случайное число:

```
random: void -> int
```

или возвращающая текущее время:

```
get_current_time: void -> Time
```

Также функция может принимать аргумент, но никак от него не зависеть:

```
foo: A a, A b -> A r
```

```
    r <- b + 1
```

Такой аргумент называется неиспользуемым (unused [ˌʌn'ju:zd]) аргументом функции. В примере это первый аргумент функции `foo` — `a`.

Занятие 8.2. Настройки Python'a

Для Py нам не нужно IDE, хватит текстового редактора с подсветкой синтаксиса. На вскидку: Kate, Geany, VSCode — хорошо настраиваются, удобные, есть плагины для Py.

- Включи показ пробельных символов. Пробелы будут в виде точки, а табы ->.
- Установи для отступов табы, а не пробелы. Для выравнивания (если есть такая настройка) - пробелы.

Размер отступа — на своё усмотрение. Стандартный — 4 символа. Я использую 8, ибо на 6-ой час программирования у меня обычно всё плывёт перед глазами. Для Py по PEP8 (нахуй PEP8) — 2 пробела, это ИМХО смерть. Потому и нужны табы: кому-то надо 2, кому-то 8, и с пробелами такое не прокатит.

- Шрифт. Я предпочитаю Terminus, и для кода, и для терминала, и он есть в репах всех дистрибутивов, оч популярный и приятный. В любом случае — это должен быть хорошо читаемый моноширинный шрифт.

Тыкал ещё Nask2, а последний год — украл какой-то Pro-шрифт для прогеров. Гугли кароч и смотри.

- Тема. Я предпочитаю мягкие светлые или темные темы. Кому-то заходят контрастные. Главное: шоб кровь с глаз не лилась, можно пробовать разные и смотреть на эффект спустя час.

Занятие 8.3. Задания

Задание:

Научитесь определять функции в Python:

- определите функцию, которая всегда возвращает 2


```
two: void → int r
      r ← 2
```
- функцию инкремента:


```
inc: int x → int r
      r ← x+1
```
- функцию сложения


```
add: int x, int y → int r
      r ← x+y
```

Научи меня:

- Можно ли объявлять функции в Python?

Занятие 10.1. Классы

▫ Класс (`class` [kla:s]). Классы — это способ описания объектов. Но если ИРЛ бытиё предшествует описанию, то в языках программирования всё наоборот: сначала идёт описание, а затем создание объектов. Такие объекты называются ▫ экземплярами класса (`class instance` [kla:s 'inst(ə)ns]).

▫ Объявление класса (`class declaration` [kla:s deklə'reɪʃ(ə)]) — это заявление о том, что существует некий класс. На нашем псевдоязыке:

```
class Apple:
```

Не сложно. ▫ Определение класса (`class definition` [kla:s defɪ'nɪʃ(ə)n]) описывает структуру объектов при помощи таких понятий как поле класса и метод класса.

▫ Поле класса (`Class field` [kla:s fi:ld]) — это некоторая переменная, являющаяся составной частью класса:

```
class Apple:
  field  Color  color
        └──┬──┘ └──┬──┘
            тип  имя поля
```

Полей может быть множество, но у каждого должно быть имя.

▫ Метод класса (`Class method` [kla:s 'methəd]) — это функция, которая имеет доступ к полям класса:

```
class Apple:
  field uint size
  get_size: Apple self → uint r
            r ← self.size
```

Первым аргументом такой функции является сам объект. Доступ к полям класса будем обозначать точкой.

Создание экземпляра класса:

```
Apple apple{}
```

для создания экземпляра класса `Apple` с именем `apple`. Доступ к методам класса обозначим точкой:

```
apple.get_size() - вызов метода у экземпляра apple
```

Здесь под точкой будет подразумеваться вызов функции `get_size` с аргументом `apple`, то есть:

```
get_size(apple)
```

Также мы не будем всегда писать первый аргумент, но всегда будем его подразумевать у методов:

```
class Apple:
  field uint size
  get_size: void → uint r
            r ← self.size
```

Экземпляр класса внутри полей будет именоваться `self`. В нашем случае `get_size` это “метод без аргументов”.

Вопросы для самоконтроля:

- + Что такое экземпляр класса?
- + К какому классу объектов вы бы отнесли Ви. Фи.?
- + Какая связь между методом и функцией?
- + Из чего состоит описание класса?
- + Что такое объявление класса и чем оно отличается от определения?

Занятие 10.2. Ввод и вывод

Ящик с дырками. У программ есть ввод и вывод. Можно представить себе программу как ящик с дырками сверху и снизу. В верхнюю дырку (ввод) можно что-то бросить. А из нижней дырки (вывод) может что-то выпасть. Внутри ящика происходят хитрые процессы, на которые может оказывать влияние ввод (а может и не оказывать). Хитрые процессы могут приводить к выводу чего-то интересного из ящика. Например, если скормить видеопроигрывателю файл с любимым фильмом, то на выходе мы получим видео и звук, которые нам нравятся.

В принципе любую программу можно свести к такому ящику. Но пока что нас будет интересовать класс программ с текстовым вводом-выводом. Примером такой программы может служить WEB-сайт, ведь он посылает в ваш браузер просто гору текста (HTML, CSS, JS кода).

▫ Стандартный вывод (Standard output ['stændəd 'aʊtpʊt]). Будем считать, что это текст, который программа выводит на экран терминала. Во многих языках программирования (и Python в их числе) для вывода в стандартный вывод надо написать что-то вроде

```
print("Привет, мир!")
```

то есть через вызов функции print с передачей ей в качестве аргумента того, что вы хотите вывести.

```
print(2), print(-666), print("K")
```

Иногда print — довольно сложная функция и умеет выводить КрАсИвО. гуглите “Python print formatted string”, хз как это гуглить на русском ааааа памагити.

Если стандартный вывод это то, что печатает в терминал программа, то ▫ стандартный ввод (standard input ['stændəd 'ɪnpʊt]) — это то, что печатает в терминал пользователь.

Со стандартного ввода значения считываются тогда, когда этого хочет программа. Это состояние называется ▫ *ожидание ввода (waiting for input)*. Чтобы программа в него перешла, надо вызвать специальную функцию. Не помню, что там в Питоне, пускай будет:

```
s = str()
input(s)
```

Здесь в качестве аргумента мы передаём переменную, в которую будет записана введенная пользователем строка. Пока пользователь не закончит ввод, программа не будет идти дальше. Обычно ожидание ввода продолжается до нажатия клавиши Ввод (Enter).

Занятие 10.3 Создание классов

Научитесь создавать классы в Python. Создайте класс:

```
class A:
    get_x: void → int r
        r ← self.x
    set_x: int x → void
        self.x = x
    field int x
```

Напишите программу на Python, создающую экземпляр класса A

```
A a{}
```

устанавливающую x равным 666 и выводящим x в консоль

```
a.set_x(666)
print(a.get_x())
```

Занятие 15.1. Деление

Деление. □ Деление (Division [di'vɪʒ(ə)n]) — это бинарная операция, обратная умножению: если $n \cdot k = m$, то $m / n = k$.

Деление с остатком. Поделим 41 на 3 в столбик:

$$\begin{array}{r} 41 \overline{) 3} \\ \underline{3} \\ 11 \\ \underline{9} \\ 2 \end{array}$$

Получилось, что 41 это $3 \cdot 13 + 2$. Число, оставшееся после деления нацело называется □ остатком (remainder [rɪ'meɪndə])

$$\begin{aligned} m &= n \cdot k + r \\ 41 &= 3 \cdot 13 + 2 \\ 8 &= 2 \cdot 4 + 0 \\ 17 &= 4 \cdot 4 + 1 \end{aligned}$$

В случае, когда $r = 0$, остаток называют □ пустым и говорят, что n делит m □ без остатка. Число n в таком случае называется □ делителем m .

Запись: $n|m$ “эн делитель эм” или “эн делит эм без остатка”. Например:

$$3|27, 2|6, 3|6, 12|144.$$

Взятие остатка от деления. В случае, когда нас интересует только остаток от деления будем писать: $m \% n = r$

Например: $41\%3=2, 5\%2=1, 6\%3=0$

Задания:

1. Поделите с остатком 40 на 6
2. $30\%7 = ?$
3. Найдите все простые делители числа 75.

Занятие 15.2. Присвоение и конструктор

Присвоение и конструктор.

□Конструктор (Constructor [kənˈstrʌktər]) класса это подпрограмма инициализации, первичной подготовки объекта (экземпляра класса). Например, беременность и роды — часть конструктора младенцев. После того, как конструктор завершит свою работу, объект готов к использованию. Конструкторов у класса может быть несколько. Например, есть способ конструирования младенцев через кесарево сечение или через частичное донашивание в инкубаторе после преждевременных родов.

До завершения работы конструктора, объект не готов к использованию. Конструктор, не выполняющий никаких действий, называется □пустым конструктором.

Конструктор может принимать параметры, аргументы и на их основе конструировать объект. Определяется конструктор обычно так же как и методы класса, но в отличие от них не может возвращать никаких значений.

На нашем псевдоязыке:

```
class Apple:
    constructor:
        self.color ← Color.RED
    constructor: Color c
        self.color ← c
    field Color color
```

класс Apple с одним полем color типа Color и двумя конструкторами: без параметра, создающий красные яблоки и с параметром c, создающий яблоки цвета c.

Запуск конструктора, как и метода, называется вызовом. Он происходит при создании экземпляра класса. Мы будем писать:

```
Apple a{Color.GREEN}
Apple b{}
```

то есть использовать {} (braces)

Следует заметить, что если конструктор класса явно не определён — это не значит, что его нет. Он подразумевается и является пустым.

▢Присвоение значения переменной меняет значение переменной. Мы будем писать ←.

Например: `var x`
сейчас значение `x` неопределено
`x ← 5`
теперь значение `x` — 5
`x ← -100`
теперь значение `x` — -100

В реальных языках программирования для вызова конструктора используются разные символы. Рассмотрим для примера C++, в котором сконструировать объект простого типа можно сразу тремя способами:

```
int x = 5
int y(5)
int z{5}
```

В результате все три переменные будут инициализированы одним и тем же значением `x = y = z = 5`.

Первый вариант выглядит так же, как присвоение в этом языке, второй — как вызов функции. Самым приемлемым является третий вариант. Впрочем, он тоже не лишён минусов.

Для инициализации “простых” типов, вроде `uint`, `char`, `enum` я буду использовать символ присвоения ←.

`int x ← 5`, `Color c ← Color.RED`, а для классов только `{ }`.


Вопросы для самоконтроля:

- + Чем отличается присвоение от конструктора?
- + Верно ли, что конструктор без параметров — это пустой конструктор?
- + Зачем нужны параметры конструктора?
- + Конструктор не смог завершить свою работу. Что с объектом?

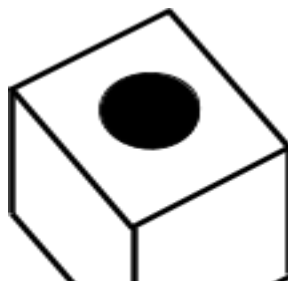
Занятие 15.3. Паттерны программирования

▢Паттерны программирования (Programming pattern ['prəʊgræm 'pæt(ə)n]).

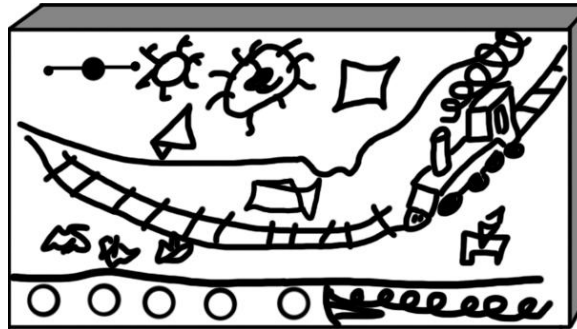
На русский переводится как “шаблоны”, но шаблоны — это `templates`, и повелось в русском называть то, о чём мы будем говорить паттернами.

Если взглянуть на ящик с дырками издали, то... 

Ну нет, поближе, конечно...



Так. Давайте сбоку и стенку прозрачную пожалст!



То можно увидеть общую структуру хитрых механизмов. Крупные, сложные структурные элементы как-то друг с другом соединены и взаимодействуют, задавая некий глобальный процесс происходящий в программе. Эта структура называется [□]архитектурой программы (program architecture ['prəʊgræm 'ɑ:kitektʃə]). Архитектура закладывается в самом начале жизни проекта и очень тяжело меняется в последствии. Разработка хорошей архитектуры — важная задача разработки серьёзных программных продуктов (ПП).

Некоторые подходы к разработке архитектуры хорошо себя зарекомендовали, имеют богатую историю и практику применения. Эти подходы называются паттернами архитектуры. Пожалуй, самым популярным является паттерн MVC — и с ним мы познакомимся поближе.

Если посмотреть в ящик при помощи лупы, то станет видно устройство всех тех кубиков — модулей — из которых состоит программа. Станут видны небольшие классы. На этом уровне накоплено большое количество опыта по решению возникающих перед разработчиком задач. Этот опыт также носит название паттернов — просто “паттерны программирования”. Но сейчас они нам не интересны и излишни, они нам не нужны.

Вопросы:

- Какова важная задача при разработке программных продуктов?

Научи меня:

- + Расскажи мне про паттерны MVC. Что вообще такое [□]Модель (model ['mɒd(ə)])? [□]Вьюха (view [vju:])? [□]Контроллер (Controller [kən'trɒlə])?
- + Какие есть сходные с MVC паттерны? В чём их отличие?

Занятие 16.1. Задания

Модифицируйте класс А из занятия 10, часть 3:

- добавьте поле `y` и методы `set_y`, `get_y` аналогично `x`.
- добавьте конструкторы

```

constructor: int x, int y
    self.x = x
    self.y = y
constructor: void
    self.constructor(0, 0)

```

Здесь второй конструктор (без аргументов) вызывает внутри себя первый с аргументами `x=0`, `y=0`.

Убедитесь, что класс работает.

Занятие 16.2. Задания

Нужна программа, выводящая три числа -1, 0, 1 на экран, через запятую. Необходимо разделить ее на три класса: View, Model, Controller.

1. Попробуйте описать, чем займутся эти классы.

Решение: [написано далее белым шрифтом]

2. Теперь опишите объявления методов, конструкторов и полей этих классов.

Возможное решение: [написано далее белым шрифтом]

class View:

```

show: int a, int b, int c → void

```

class Model:

```

int a
int b
int c

```

class Controller:

```

constructor: View v, Model m
show numbers: void → void

```

3. Напишите эту программу на Python в файле `show_num.py`.

МОДУЛЬ 2

В этом модуле мы познаем Истину и Ложь, познакомимся с упорядоченной парой, узнаем о способе представления алгоритмов в виде схемы. Самое главное: вы познакомитесь с техническим заданием на ваш первый проект, игру крестики-нолики!

Занятие 20.1. Свёртка и упорядоченные пары

Введем следующий способ определения множеств — \square свёртку (set comprehension): $\{\textcircled{1} \mid \textcircled{2}\}$

Здесь на месте $\textcircled{1}$ записывается общий элемент множества, а на $\textcircled{2}$ — условие, которое выполняется для всех элементов множества.

Например: $\{x \mid x < 3 \text{ и } x \in \mathbb{N}\}$ это $\{0, 1, 2\}$. Читается: "множество всех x , для которых x — натуральное число меньше трёх."

В множестве порядок не важен. Это значит, что $\{\text{foo}, \text{bar}\}$ это то же самое, что и $\{\text{bar}, \text{foo}\}$.

\square Упорядоченную пару (Ordered ['ɔ:dəd] Pair [реэ]) мы будем обозначать в угловых скобках и определим так: $\langle \text{foo}, \text{bar} \rangle$ это $\{\{\text{foo}\}, \{\text{foo}, \text{bar}\}\}$.

Главное свойство упорядоченной пары таково:

Если $\langle a, b \rangle = \langle c, d \rangle$, значит $a = c$, $b = d$.

Аналогично определим упорядоченную тройку и так далее, упорядоченную n -ку:

$$\langle a, b, c \rangle = \langle \langle a, b \rangle, c \rangle$$

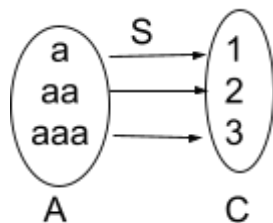
$$\langle a_1, a_2, \dots, a_{n-1}, a_n \rangle = \langle \langle a_1, \dots, a_{n-1} \rangle, a_n \rangle \text{ — упорядоченная } n\text{-ка}$$

\square Декартовым произведением (Cartesian [ka:'ti:ziən] Product ['prɒdʌkt]) x множеств X и Y называется Множество всех упорядоченных пар $\langle x, y \rangle$, таких, что $x \in X$, $y \in Y$:

$$X \times Y = \{\langle a, b \rangle \mid a \in X, b \in Y\}$$

Читается: "Декартово произведение x на y есть все упорядоченные пары a - b , где " a " из (принадлежит) x , и " b " из (принадлежит) y ".

Любое подмножество $X \times Y$ есть отношение. Например:



$$A = \{a, aa, aaa\} \quad C = \{1, 2, 3\}$$

$$S = \{\langle a, 1 \rangle, \langle aa, 2 \rangle, \langle aaa, 3 \rangle\}$$

$$A \times C = \{\langle a, 1 \rangle, \langle a, 2 \rangle, \langle a, 3 \rangle, \langle aa, 1 \rangle, \langle aa, 2 \rangle, \langle aa, 3 \rangle, \langle aaa, 1 \rangle, \langle aaa, 2 \rangle, \langle aaa, 3 \rangle\}$$

Очевидно, что $S \subseteq A \times C$

пример 0.1

Вопросы

- Как читается:
 - $z \in A$
 - $A \subseteq B$
 - $C \subset D$
 - $a \in \langle a, b \rangle$, $a \in X$, $b \in X$, $a \neq b$
 - $\{\langle a, b, c \rangle \mid a, b, c \in \mathbb{Z}, c = a + b\}$

- Как будет выглядеть функция возведения в куб из Множества $\{0, 2, 3, 4\}$ в Множество $\{40, 2, 3, 1, 4, 10, 16, 9, 7, 8, 27\}$?
- Какими свойствами обладает отношение S из последнего примера (пример 0.1)?

Занятие 20.2. Прелести и радости. Абстрактное “Мяу”

Множество \mathbb{Q} рациональных чисел (Rational [ˈrɑːʃ(ə)n(ə)l] Number [ˈnʌmbə]) (от латинского ratio — отношение) состоит из \mathbb{Z} и дробных чисел. Обозначается \mathbb{Q} .

Любое рациональное число $a \in \mathbb{Q}$ можно представить в виде дроби $\frac{m}{n}$, где $m \in \mathbb{Z}$, $n \in \mathbb{N}$ и $n \neq 0$.

$$\mathbb{Q} = \{x \mid x = m/n, m \in \mathbb{Z}, n \in \mathbb{N}, n \neq 0\}$$

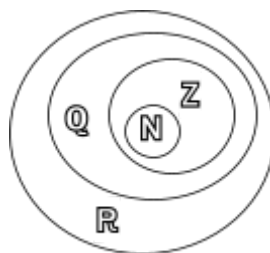
Каждому $a \in \mathbb{Q}$ соответствует единственная точка на координатной прямой, но есть точки прямой, которым не соответствует ни одна $a \in \mathbb{Q}$.

\mathbb{I} Иррациональные числа (Irrational [ˈɪrəʃ(ə)n(ə)l] Number [ˈnʌmbə]) — это числа, которые могут быть представлены в виде бесконечной непериодической дроби. Например, $\sqrt{2}$, π , e и тому подобное.

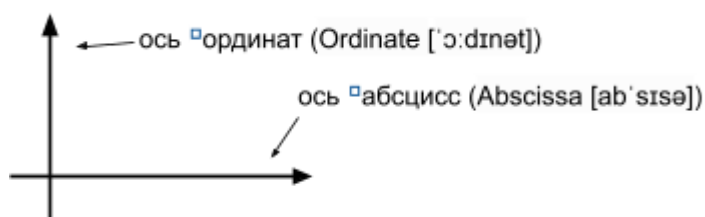
\mathbb{R} Множество вещественных (действительных) чисел (set of real [rɪəl] numbers) \mathbb{R} состоит из \mathbb{Q} и иррациональных чисел. Каждой точке на координатной прямой соответствует некоторое вещественное число:



$$\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$$

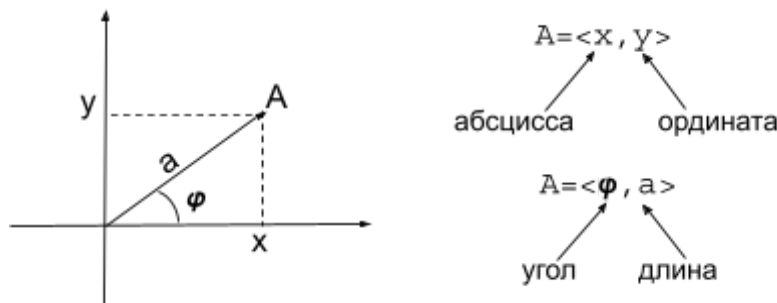


Если координатная прямая — это способ представления множества \mathbb{R} , то координатная плоскость — $\mathbb{R} \times \mathbb{R}$



\mathbb{S} Скаляр (Scalar [ˈskeɪlə]) — величина, которая может быть представлена одним числом. Например, рост, количество звёзд на небе.

▫ Вектор (Vector ['vɛktə]) — величина, которая представляется несколькими числами. Например, точка на плоскости: $\langle x, y \rangle \in \mathbb{R} \times \mathbb{R}$. Стоит отметить, что точка на плоскости может быть представлена не только двумя координатами, но и углом и длиной вектора:



▫ Функция от двух переменных (бинарная функция) — это функция, отображающая декартово произведение двух множеств на третье множество: $f_{\circ\circ}: X \times Y \rightarrow Z$, то есть функция, принимающая уп.пару $\langle x, y \rangle \in X \times Y$ в качестве аргумента.

Аналогично мы можем записать $Z = f_{\circ\circ}(x, y)$, $x \in X, y \in Y, z \in Z$. Или, как я описываю в псевдокоде: $f_{\circ\circ}: X \ x, \ Y \ y \rightarrow Z \ z$

Тем же образом определяются функции от 3-х, 4-х и так далее аргументов ($\text{bar}: X \times Y \times Z \rightarrow A$).

Здесь $f_{\circ\circ}(x, y)$ означает не что иное как $f_{\circ\circ}(\langle x, y \rangle)$.

▫ Суперпозицией (композицией) \circ двух отношений g и f ($g \circ f$) называется отношение из $D(f)$ в $R(g)$ такое что если $\langle x, y \rangle \in f$, то $\langle y, z \rangle \in g$.

$$g \circ f = \{ \langle x, z \rangle \mid \langle x, z \rangle \in D(f) \times R(g) \text{ и } \langle x, y \rangle \in f \text{ и } \langle y, z \rangle \in g \}$$

Суперпозиция двух функций является функцией. Классическая запись суперпозиций двух функций h и l :

$$(h \circ l)(x) = h(l(x))$$

Если значением функции является вектор

$$f_{\circ\circ}: X \rightarrow A \times B \times \dots \times Y, \text{ то есть}$$

$$f_{\circ\circ}(x) = \langle a, b, \dots, y \rangle, \text{ где } x \in X, a \in A, b \in B \text{ и}$$

так

далее, то такая функция может быть представлена как упорядоченная n-ка

▫ компонентных (Component [kəm'pəʊnənt]) функций.

$f \circ \circ (x) = \langle f_a(x), f_b(x), \dots, f_y(x) \rangle$, где
 $f_a(x) = a$ и так далее.

читается "эф а'тое от икс равно а"

f_b "эф бэ'тое"

□ индекс (Index ['index])

Научи меня:

- Свойства чисел из \mathbb{N} , \mathbb{Z} , \mathbb{Q} , \mathbb{R}
- Что такое десятичная дробь?
- Свойство плотности \mathbb{Q} .
- Что такое рациональное приближение действительного числа?
- Свойства суперпозиции отношений.
- Какая функция называется биекцией?
- Мощность Множества.
- Что такое □ синглтон (Singleton ['sɪŋɡ(ə)lt(ə)n])
 - в матеше
 - в погромировании

Вопросы:

- Приведи пример числа из \mathbb{R} . Верно ли, что $2 \in \mathbb{R}$?
- Что такое скаляр? Пример?
- Точка A в трёхмерном пространстве выражена тремя числами x_A , y_A , z_A .
Как бы записать точку A?
- Параметры насоса указываются следующим набором: скорость прокачки (до 1000 м³/с); *напряжение питания (до 360 В); *переменный или постоянный ток питания; * мощность (до 100 кВт); *наличие встроенного регулятора; *рабочая температура (от -40°C до +50°C). Определите множества для всех параметров насоса. Определите свёрткой множество всех возможных параметров насосов.
- $f(x) = x+1$, $g(x) = x^2$.
 $f \circ g = ?$ $g \circ f = ?$

Занятие 20.3. Нейминг

Нейминг

ЭтоКэмелКейс CamelCase

а_это_снейк_кейс

естьИДругие Всякие_Варианты

Вопрос такой: CamelCase или snake_case? Ответ: чё больше нрав.

В компании Microsoft эти инвалиды добавляют к именам переменных тип что-то вроде “i_foo” (i — integer) для чисел. Типа шоб было видно, какого типа переменная. Это кал. Проверкой типов занимается компилятор, а имя класса от имени переменной должно отличаться цветом, которым выделяет подсветка синтаксиса.

Я обычно называю классы КэмелКейсом, методы тоже, но с маленькой.

```
class FooBar:
    isfoo: void → bool
    add: int → void
```

А поля — snake_case-ом:

```
field int sum_total
```

Главное — единообразиие. В рамках одного проекта правила должны быть одинаковы. Если решено именовать классы snake_case — во всём проекте должно быть class foo_bar

Поэтому с этим моментом надо определяться в самом начале любого проекта.

Все, я закончил 12.-5.2020

А, нет. Я к приватным полям добавляю нижний пробел. Справа или слева не помню.

Ах да, константы.

КОНСТАНТЫ_ПИШУТСЯ_СНЕЙК_КЕЙСОМ_КАПСОМ (uppercase)

Занятие 21. Истина и Ложь

Тип \square boolean (логический, bool [bu:l]).

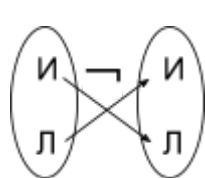
bool = {Истина, Ложь}

Всего два значения. Кодировются по-разному. Например И и Л, True и False, Т и F, true и false, 1 и 0.

► Правило: всё, что не Ложь — это Истина.◀

\square Одноместная функция (one-place [wɒn-plɛɪs] function)— значит один аргумент

Одноместная функция “НЕ”



\neg : bool→bool

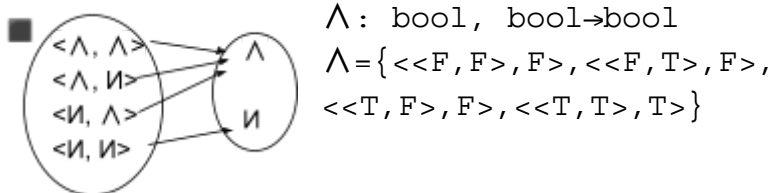
$\neg = \{<0, 1>, <1, 0>\}$

\neg — это отрицание

Не Истина — Ложь. А Не Ложь — Истина. $\neg 0=1$, $\neg 1=0$



Бинарная операция “И” — \square конъюнкция (Conjunction [kən'dʒʌŋ(k)](ə)n])



Конъюнкция принимает значение "Истина" только в том случае, если оба операнда (Operand ['орэ́нд]) в значении "Истина". Во всех других случаях значение \wedge — "Ложь". Пример: Витя классный и сейчас дождь. Первую часть обозначим A , вторую B . $A \wedge B$. Витя действительно классный, значит A — истинно. Но дождя за окном не наблюдается, значит B — ложно, а поэтому все выражение $A \wedge B$ — ложно. Примером истинного выражение может быть такое: Крокодилы и ёлки зелёные.

	\neg
0	1
1	0

$\neg: \text{bool} \rightarrow \text{bool}$

Введем ещё один способ задания функции — таблица. Здесь первый столбец содержит входные данные, а второй — значение операции \neg .

Для конъюнкции:

Пример для отрицания: Витя не красавчик ($\neg A$)

Очевидно — это ложь, так как A — истинно.

	\wedge
0 0	0
0 1	0
1 0	0
1 1	1

$\wedge: \text{bool} \times \text{bool} \rightarrow \text{bool}$

Научи меня:

- Расскажи про операции
 - дизъюнкция
 - исключающее ИЛИ

Построй для них таблицы значений.

- Кто за Джордж Буль?
- Какие интересные способы обзывания классов / функций / методов / переменных / полей / констант есть?

Вопросы:

- Что такое Истина?
- $\neg \subseteq ?$
- $D(\wedge) = ?$
- $((1 \wedge 0) \vee 1) \wedge 0 = ?$
- Приведи пример для выражения: $(\neg A) \wedge (\neg B) \wedge (C \vee D)$

Занятие 22. Задания

A. Написать базовый класс:

```
class Base:
  -baseMethod:
    print("I am base Method.")
  -field String baseField="I am base Field."
```

Затем его потомка:

```
class Child0:parent Base
  -child0Method:
    print("I am child0Method.")
  -field String child0Field="I am child0Field."
```

Аналогично образовать потомка от Child0 с именем Child1. Создать по экземпляру каждого класса, убедиться, что экземпляр Child0 имеет поле baseField и метод baseMethod, аналогично для Child1: поля и методы Child0 и базового класса.

B. Реализовать функцию

```
foo: bool a, bool b, bool c → bool ret
ret =  $\neg a \wedge (b \vee \neg c)$ 
```

вывести на экран все входные значения и значение функции foo на этих аргументах в четырех столбцах. Вместо 0/1 можно использовать любую другую кодировку логических значений.

abc	ret	
000	1	
001	.	
010	.	R(foo)=? D(foo)=?
011	.	
10.	.	
1..	.	
...	.	
..	.	
.	.	

C. Написать четыре функции:

Инкремента:

```
increment: int x → int ret
ret = x + 1
power2: int x → int ret (квадрата)
ret = x * x
```

их суперпозиции $\text{inc} \circ \text{pow2}$

```
incpow2: int → int
```

и $\text{pow2} \circ \text{inc}$

```
pow2inc: int → int
```

Убедиться, что на x от 0 до 5 значения `incrow2(x)` равны `increment(power2(x))` и `pow2ink(x)` равны `power2(increment(x))`.

D. Написать класс

```
class Pair:
    field A first           ← 1-ый элемент пары
    field B second         ← 2-ый элемент пары
    - constructor:         ← пустой конструктор
                           (first=second=None)
    - constructor: A, B    ← устанавливают значения
    - set: A, B → void     ↓
    - clear:               ← очищают (f=s=None)
    - is Empty: void → bool ← пусто? True если и first, и second
                           пусты (равны None)
    - compare: Pair → bool ← сравнивает self с другой парой,
                           истина если self.first = другой.first
                           и second тоже. AAA, похоже если
                           пары равны ⇒ Истина
    - copy: void → Pair   ← конструирует новый экземпляр
                           Pair с такими же значения и его
                           возвращает.
```

Убедиться, что `Pair` работает как надо.

Хочу сказать, что я не тороплю с выполнением работы. Каждую часть можно делать по 3 дня: в первый — теорию, во второй — Научи меня, и вопросы — в третий. Практические задания рекомендую выполнять отдельно, между или после частей — когда теория уложится в голове. Если нет понимания — ничего страшного: бездумное применение заученных правил тоже идёт в зачёт.

Конечно же я не запрещаю приступать к чему-либо в любом порядке.

Занятие 23.1. Английский язык

Каков уровень вашего английского? Вам необходимо освоить свободное чтение технической документации. Для этого необходимы базовые знания грамматики.

- Чтение слов и транскрипций
- Побудительные предложения
- Род и число имен существительных, личные местоимения
- Спряжение `to be` по временам
- Артикли

- Указательные местоимения
- Предлоги мест и направления
- Притяжательные местоимения
- Времена группы Simple
- The Present Continuous Tense
- Отношение родительного падежа в англ.
- Прямое и косвенное дополнения
- Числительные, порядковые числительные
- Модальные глаголы can и must, обороты to be able to и to have to
- Participle II
- The Present Perfect Tense
- The Passive Voice
- Словообразование, суффиксы и префиксы

I'm sure it will be easy for you to learn all that.

В интернете есть хорошие курсы с игровой формой обучения. Не поспкутитеся тысячи рублей, не пожалейте времени. Занимайтесь регулярно, лучше каждый день понемногу.

Занятие 23.2. Как заниматься #2

Оцените, как успешно вы занимаетесь. Быстро ли вы осваиваете новый материал? Удаётся ли запомнить много нового?

Если что-то не получается, то вот ещё несколько советов:

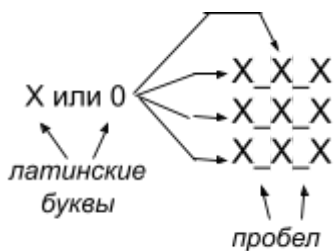
- Пишите. Рукой, конспекты. Это лучше, чем печатать. Даже если вы никогда в жизни не будете их перечитывать. Пишите своими словами. Ответы на вопросы можно слать мне рукописные! (Если ваш почерк читаем...)
- Говорите, делитесь новой информацией. Можно привлечь партнёра и попытаться объяснить ему материал.
- Загуглите информацию, посмотрите ролик по теме на YouTube, почитайте статью в Wikipedia.
- Советую книгу Барбары Оакли "Думай, как математик". Возможно имеет смысл идти по ней параллельно с занятиями.

Помните, что главное — ваша мотивация. Если сейчас вам просто интересно, то это когда-нибудь может закончиться. Вам стоит заранее подумать, почему вы хотите освоить разработку программного обеспечения, и двигаться к этой цели.

Занятие 26. Техническое задание на проект 'хо' (версия 0.1)

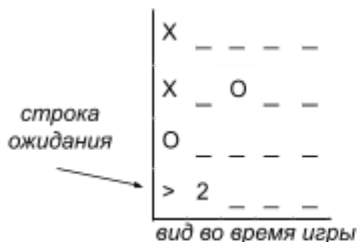
Игра в крестики-нолики

Начинается сразу после старта программы. Ходят игроки по очереди, сначала запрашивается ввод у крестиков, затем у ноликов.



Поле: 5 символов в ширину, 3 в высоту. Ввод - число от 1 до 9 включительно, что означает ячейку, в которую игрок хочет поставить свою отметку. Ячейки нумеруются слева-направо сверху-вниз.

После того, как игрок выстроил 3 свои метки в линию, он объявляется победителем. Новая игра начинается нажатием кнопки "Ввод".



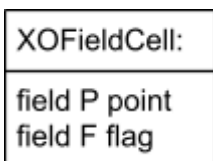
Любой ввод, кроме 1-9, должен игнорироваться.

△Конец. Всё. ••

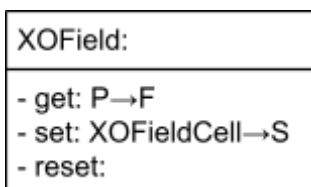
Не торопись приступить к реализации. Техническое задание (ТЗ) — это ответ на вопрос "как результат должен выглядеть снаружи?". Нам необходимо решить, как он будет выглядеть внутри. И, зачастую, это непростой вопрос. И очень важный.

Мы будем эксплуатировать паттерн MVC. Модель, рулящая хранением данных, Контроллер с основной логикой и Вьюха, выводящая информацию пользователю.

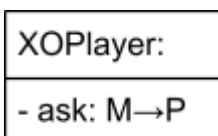
С моделью все просто: есть поле игровое, есть игроки. С полем тоже всё, вроде, ясно: на поле девять клеток, которые могут быть пусты или с крестиком или с ноликом.



← Это класс, описывающий клетку поля. Он прост: содержит только два публичных поля: point для указания места этой клетки на поле и flag для хранения флага (нет флага X или O).



← Игровое поле может сообщать состояние. Его можно менять, передав XOFieldCell с указанием клетки и нового флага.



← Игрока можно спросить куда он ходит, передав ему состояние поля и получив в ответ указание на клетку поля. Не придумал ничего лучше ask. Может быть стоит назвать метод Turn? Или как ещё?

XOView:
print_field: ? → void
print_winner: ? → void

← В Техническом задании требуется примитивный консольный интерфейс — будем делать примитивную вьюху.

XOController:
- constructor: XOField, XOView,XOPlayer,XOPlayer
- play

← Контроллер, принимающий модели, вьюху и имеющий один метод для процесса игры.

Есть следующие основные нерешенные проблемы:

- Тип P: чем кодировать клетки поля? Беззнаковым целым от 1 до 9 как в Техническом задании или переводить в координаты $\langle x,y \rangle$, где $x,y \in \{0,1,2\}$? Почему?
- Тип F: чем кодировать состояние клетки? Почему?
- Нужен ли тут вообще отдельный класс XOFieldCell или это лишнее усложнение? Почему?
- Кто должен следить за правильностью выбора клетки, проверять, не пытается ли игрок поставить на занятое поле свой знак (флаг) или что вообще в модель передаётся не тот флаг? XOField, XOPlayer или XOController? Почему? Что тогда должен возвращать XOField.set, то есть тип S — что это?
- Кто будет проверять, что получилось три в линию? Почему?
- Каким образом передавать в XOPlayer.ask состояние поля чтоб не проебать инкапсуляцию, то есть тип M это что? Почему? А если так, то может быть что-то надо изменить?
- Нужно ли что-то менять в XOView? Почему?
- Короче, необходимо модифицировать схему как-то. А также решить, заводить ли ещё класс XO где всё это будет скрепляться или сделать всё в `__init__`?

Какие вообще идеи по Техническому заданию? Всё ли ясно или что-то пропущено?

Занятие 30.1. Ветвление

Ветвление. Условный оператор *если...то...иначе*.

Условный оператор принимает один аргумент. Если он истинен, то выполняет код, указанный после *то*:

```
if (n ≠ 0)
then:
    x ← m/n
```

Этот пример демонстрирует проверку знаменателя на 0 перед выполнением деления. Деление будет произведено только в том случае, если $n \neq 0$ истинно, то есть значение n не равно нулю.

Если необходимо выполнить другой код в случае, когда аргумент `if` ложен, используется конструкция *if...then...else*:

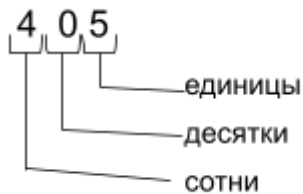
```
if (n ≠ 0)
then:
    x ← m/n
else:
    x ← 0
или
```

```
if (n = 0)      если
then:          то
    x ← 0
else:         иначе
    x ← m/n
```

что одно и то же, но первый вариант выглядит понятнее. Рекомендуется строить \square условие (conditions [kən'diʃənz]) (аргумент оператора if) так, чтобы исход Истина был вероятнее, то есть код после then выполняется чаще.

Занятие 30.2. Системы счисления

\square Десятичная система счисления (СС) (Decimal number system ['desɪm(ə)'nʌmbə'sɪstəm]). Это самая привычная нам система счисления, ибо у нас 10 пальцев. В ней имеется 10 \square цифр (цифра - numeral: {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}). Число представляет собой \square разряды (Digit ['dɪdʒɪt]), состоящие из цифр. Номер разряда считается справа налево.



В нашем числе 405 пять единиц, ноль десятков и четыре сотни.

Итоговое число считается так:

$$\text{в 1-м разряде } 5 \Rightarrow 5 \cdot 10^0,$$

$$\text{во 2-м } 0 \Rightarrow 0 \cdot 10^1,$$

$$\text{в 3-м } 4 \Rightarrow 4 \cdot 10^2.$$

Итого:

$$5 \cdot 10^0 + 0 \cdot 10^1 + 4 \cdot 10^2 = 5 \cdot 1 + 0 \cdot 10 + 4 \cdot 100 = 5 + 0 + 400$$

Всё это кажется очевидным, но это потому, что мы привыкли к этому.

Теперь надо привыкнуть кое-к чему ещё

\square Двоичная система счисления. В ней всего две цифры: {0, 1}, а каждый разряд означает степень двойки.

Принцип тот же самый: умножаем цифру из разряда на соответствующую степень двойки: $1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1 \cdot 8 + 0 + 2 + 1 = 11$

1	0	1	1
2^3	2^2	2^1	2^0
8	4	2	1

Для ясности перед двоичными числами будем писать букву *b*: $b1011 = 11$

Для обратного перевода, из десятичной в двоичную, нужно просто последовательно делить на два:

$11 / 2 = 5 + 1$	$19 / 2 = 9 + 1$	} $19 = b10011$
$5 / 2 = 2 + 1$	$9 / 2 = 4 + 1$	
$2 / 2 = 1 + 0$	$4 / 2 = 2 + 0$	
$11 = b1011$	$2 / 2 = 1 + 0$	

$32 / 2 = 16 + 0$	} $32 = b100000$
$16 / 2 = 8 + 0$	
$8 / 2 = 4 + 0$	
$4 / 2 = 2 + 0$	
$2 / 2 = 1 + 0$	

Для устного перевода мне больше нравится такой способ:
 Переводим 19. Ищем ближайшую степень 2-ки < 19 . Это $16 = 2^4 \Rightarrow$ 5-й разряд 1.
 Далее, наименьшая степени 2-ки $< (19-16=3)$. Это $2 = 2^1 \Rightarrow$ 2-й разряд 1. Теперь
 $< (3-2=1)$, это $1 = 2^0 \Rightarrow$ 1-й разряд 1. Все разряды между ними это 0:

$$\begin{array}{r} 1\ 0\ 0\ 1\ 1 \\ 16\ 8\ 4\ 2\ 1 \\ 16+0+0+2+1 = 19 \end{array}$$

Занятие 30.3. Задания

Разработайте функцию \square деления (Division [di'viʒ(ə)n]) `div`, принимающую `Pair` (Занятие 20.3, задание D) из двух чисел и возвращающую число-результат деления ("частное от деления") `first` на `second` $\neq 0$, (если 0, то деление не должно происходить, а результатом функции `div` в этом случае должен быть ноль).

Убедитесь, что функция работает.

Занятие 31.1. Задания

Поделите на 0 в Python. Попробуйте разобраться, что произошло.

Занятие 31.2. Задания

- Переведите из двоичной системы счисления (CC2, bin) в десятичную систему счисления (CC10, dec):
 - b10
 - b1010
 - b11110000
 - b1000'0110
- Переведите из десятичной системы счисления (CC10, dec) в двоичную систему счисления (CC2, bin):
 - 10
 - 255
 - 128
 - 49

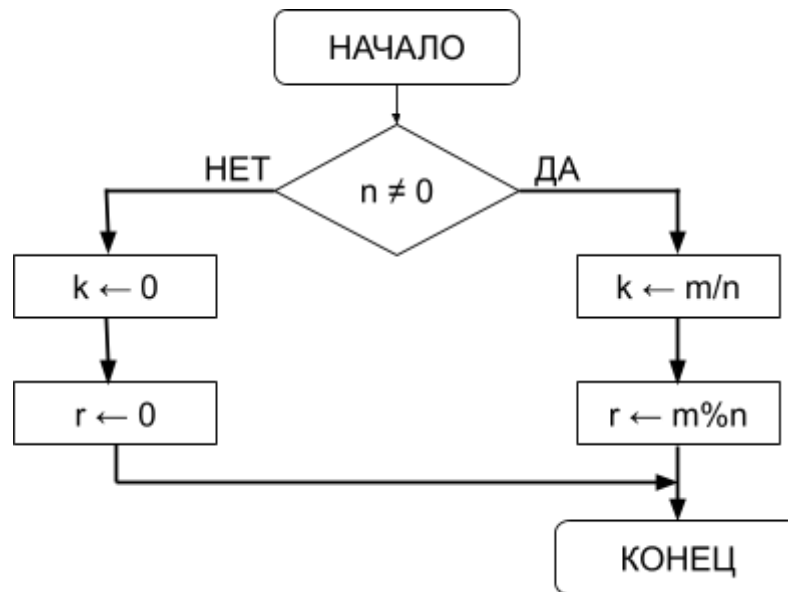
Занятие 35.2. Схемы алгоритмов

Познакомимся с очень простым и наглядным способом графического изображения алгоритмов. В нём алгоритм изображается в виде графа с четырьмя основными типами вершин:



Стрелочка указывает направление движения, которое всегда от начала к концу. Условная вершина реализует ветвление алгоритма.

Попробуем изобразить деление m на n и запись результата в переменные k и r :



Я думаю, что всё должно быть интуитивно понятно.

Начальной и конечной вершин всегда по одной: одно начало, один конец.

Вопросы для самоконтроля:

- + В каком направлении идёт движение по схеме?
- + Какие основные типы вершин существуют?

Занятие 35.3. Задания

Разработайте метод класса `Pair` `mul` перемножающий между собой *first* и *second* элементы `Pair`.

Используя методы `mul` и `div` в качестве компонентных функций, разработайте метод `muldiv`: $Pair \rightarrow Pair$, результатом которой являются пара чисел: произведение и частное от деления.

Убедитесь в корректной работе.

Занятие 36.1. Задания

Перевод предложений русского языка на язык символов.

Буду просто приводить примеры, оке?

- Я прав или он. $A \vee B$
- Его толкнули на землю и заломили руку. $A \wedge B$
- Ты не прав, он сам упал. $\neg A \wedge B$
- Я вчера его встретил, окликнул его, но он не обернулся. $A \wedge B \wedge \neg C$
- Он не услышал или не хотел с тобой встречаться? $\neg A \vee \neg B$

- Я не знаю. $\neg A$

Занятие 36.2. Задания

1. Переведите на язык символов:
 - У мыши четыре лапы и у человека тоже.
 - Я украл, выпил и сел в тюрьму.
 - Я не должен это делать.
 - Это либо супермен, либо самолёт.
 - Он не врёт, он не знает.
2. Приведите пример для следующей формулы:
 - $\neg A \vee \neg B$
 - $A \wedge B \wedge C$
 - $\neg(A \vee B) \vee C$

Занятие 36.3. Чётность-нечётность

Чётность-нечётность (even-parity or odd-parity).

Натуральное число считается \square чётным (even numbers ['i:vn nлmbəz]), если оно без остатка делится на два, то есть 2 является делителем этого числа. Число 0 так же является четным. Всё остальное — \square нечётные (odd numbers).

Задание: Разработайте функцию, которая определяет чётность-нечетность. Напишите её на Python, оформив в виде отдельного модуля (файла.py), импортируйте её из модуля и убедитесь в её работоспособности.

МОДУЛЬ 3

В этом модуле мы доберем знания, необходимые для реализации проекта 'хо', о работе с массивами и интерфейсах. Также мы углубим наши знания математики и начнем говорить о работе компьютера.

Занятие 40.1. Шаблоны

\square Шаблоны (templates ['templɪts]). Часто одна и та же по смыслу функция может быть использована с разными типами. Например, возводить в куб можно и натуральные числа, и любые целые, и так далее. Чтобы не писать для каждого типа отдельную функцию, введём понятие шаблонного типа — такого типа, который может быть разным. Обозначать будем одной буквой, начиная с T: T, U, V, ... Намеример:

```
power3: T x → T r
      r ← x • x • x
```

Такая запись будет означать, что `r` и `x` одного и того же типа. Функция с шаблонным типом — это шаблонная функция.

При вызове такой функции будем указывать конкретное значение шаблонного типа в угловых скобках: `power3<uint>(2)` для возведения в куб беззнакового целого.

Я языках программирования с динамической типизацией ничего указывать не надо, а все функции фактически являются шаблонными.

Аналогично с шаблонными классами:

```
class Apple:
    field T position
    field U color
    get_position: void → T
    get_color: void → U
    set_color: U → void
```

И для создания экземпляра шаблонного класса:

```
Apple <int, Color> a{}
```

а с типом `T=int` и `U=Color`

Занятие 40.2. О наследовании

Внятно о наследовании.

Класс — это описание совокупности объектов, обладающих некими общими свойствами (Property ['propɛti]). По этому описанию создаются конкретные объекты — экземпляры класса.

Механизм наследования позволяет описывать новые классы на основе уже существующих. Класс объявляется родителем нового класса, все свойства родителя достаются потомку.

У класса может быть несколько родителей — тогда говорят о множественном наследовании (Multiple ['mʌltɪp(ə)l] Inheritance [ɪn'hɛrɪt(ə)ns]).

Класс, не имеющий родителей называется базовым классом (Base [beɪs] Class [kla:s]).

Дальше речь пойдёт о публичных методах (Public ['pʌblɪk] Method ['mɛθəd]) и полях.

Объявим базовый класс MyServer, который умеет говорить свою версию

MyServer:
- get Home Page: void→String
- version: void → String

и домашнюю страницу. Приходит котшка Пуся и говорит: “Какой хороший сервер! Только мне нужна ещё страница обратной связи.” А нам она не нужна — и никакого мусора в уютный тёплый MyServer мы тащить не

намерены.

Тогда Пуся объявляет свой PussyServer потомком MyServer и получает

PussyServer: parent MyServer
- getContactPage: void → String
- getBlackJack: void → Blackjack

реализацию домашней страницы автоматически. Теперь, когда мы улучшим MyServer, добавив GIF анимацию на HomePage, Пуся будет рада, не приложив никаких усилий.

Разумеется, что пулемётчик Семён может продолжить цепочку, унаследовав свой сервер от PussyServer.

Научи меня:

- ❑ Расскажи о наследовании в Python. Есть ли в Python множественное наследование?

Занятие 40.3. Книга “Графы и их применение”

Найдите замечательную книжку Ойстина Оре “Графы и их применение”. Вдумчиво читайте по параграфу в день и выполняйте задания. (До просветления.) (В конце книги есть ответы.)

Осторожно! Греческие буквы! Если видите непонятную загогулину — скорее всего это греческая буква. Например ρ — читается “ро”. И в формулах $\rho(A_n)$ — “ро от \acute{a} \acute{e} нное” или просто “ро \acute{a} \acute{e} н”. Некоторые буквы, например Σ (сигма большое), имеют особое значение — гуглите. A' — “ \acute{a} штрих”, A'' — “ \acute{a} два штриха”, ρ^* — “ро звёздочка”.

В книге 37 параграфов. Если вы осилите ее за 2 месяца — отличный результат. Не стоит слишком затягивать, больше 3 месяцев — уже перебор.

Занятие 43.1. Импликация

□Импликация (Implication [ɪmˈplɪˈkeɪʃ(ə)n]) (следование) \rightarrow .

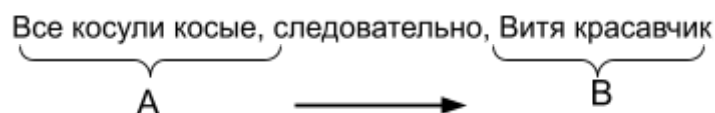
Таблица истинности для импликации:

	\rightarrow
00	1
01	1
10	0
11	1

Словами можно определить импликацию так:

1. Из лжи может следовать всё, что угодно ($L \rightarrow L=I$, $L \rightarrow I=I$).
2. Из истины может следовать только истина ($I \rightarrow L=L$, $I \rightarrow I=I$)

Пример:



A, очевидно, ложно, B истинно, в целом всё выражение истинно. Если слева от импликации Ложь — справа можно уже не смотреть, так как всё выражение истинно.

Пример ложного выражения:

(Если) обычно мыши с хвостом, следовательно (то, значит, получается), хвостатая кошка — это мышь. ($I \rightarrow L$), в целом всё выражение ложно.

Задание:

Переведите на язык символов:

- Сижу пержу.
- Моя мама белая, мой батя белый, следовательно, я тоже белый.
- Ваза разбита, значит или ты её разбила, или ты разбил.
- Он не успел, потому и не доволен.

Занятие 43.2. Таблица истинности функций

Будем коротко записывать таблицу истинности функций так:

$$S(\rightarrow) = \langle 1101 \rangle, \quad S(\wedge) = \langle 0001 \rangle, \quad \text{то есть в виде вектора значений}$$

функции на всех входных значениях, начиная с $\langle 0, 0 \rangle$:

$$S(\rightarrow) = \langle \rightarrow(0, 0), \rightarrow(0, 1), \rightarrow(1, 0), \rightarrow(1, 1) \rangle$$

Для отрицания: $S(\neg) = \langle 10 \rangle$

При построении таблицы истинности удобно вычислять значения поэтапно. Рассмотрим функцию **foo** от трёх переменных x_1, x_2, x_3 :

$$\text{foo}(x_1, x_2, x_3) = (\neg x_1) \wedge x_2 \oplus (x_2 \rightarrow x_3) \oplus x_3$$

Определим порядок вычисления: 2 3 4 1 5

Теперь последовательно вычислим операции одну за другой:

x_1	x_2	x_3	1 $x_2 \rightarrow x_3$	2 $\neg x_1$	3 $2 \wedge x_2$	4 $3 \oplus 1$	5 (или foo)
0	0	0	1	1	0	1	1
0	0	1	1	1	0	1	0
0	1	0	0	1	1	1	1
0	1	1	1	1	1	0	1
1	0	0	1	0	0	1	1
1	0	1	1	0	0	1	0
1	1	0	1	0	0	1	1
1	1	1	1	0	0	1	0

Тогда $S(\text{foo})$ это: $S(\text{foo}) = \langle 10111010 \rangle$

P.S. Здесь \oplus — это исключающее или (XOR).

Занятие 43.3. Массив данных

▣ Массив (Array [ə'reɪ]) данных. Это упорядоченное множество однообразных (однотипных) элементов, обычно расположенных в памяти один за другим. Доступ к элементам массива осуществляется по их индексу. Индекс — это натуральное число. Обычно индекс первого элемента в массиве — 0. Массивы бывают двух видов: ▣ статические (static ['stætɪk] array), размер которых фиксирован при создании, и ▣ динамические (dynamic [dæɪ'næmɪk] array), размер которых может меняться во время работы.

Рассмотрим вектор:

$$V = \langle -3, 1, 4, -3, 1 \rangle$$

Пронумеруем элементы вектора слева направо v_0, \dots, v_4 . Каждый элемент вектора $v_i \in V$, $i = 0 \dots 4$ (“вэ и́тое из вэ-большое и́ от нуля до четырёх”) это целое число: $v_i \in \mathbb{Z}$

Аналогичный этому вектору статический массив будет записываться так:

$$\text{Array}\langle \text{int}, 5 \rangle V\{\langle -3, 1, 4, -3, 1 \rangle\}$$

Здесь $\langle \text{int}, 5 \rangle$ означает тип элементов и ▣ длину (длина - Length [len(k)θ]) (▣ размер (size [saɪz])) массива.

Размер массива можно узнать, вызвав метод size: $V.size()$

Получить доступ ко второму элементу:

$$V[1]$$

то есть при помощи оператора ▣ *квадратные скобочки* (Square Brackets [skweə 'brækɪts]), передав индекс элемента.

А какой элемент массива чётный, а какой нечётный? Смотря, как их считать.

$$A = \langle a_0, a_1, \dots, a_{n-1} \rangle$$

$$B = \langle b_1, b_2, \dots, b_m \rangle$$

В случае A, чётные — это a_0, a_2, a_4 и так далее, в случае B — b_2, b_4, b_6 и так далее.

Будем придерживаться такого варианта: нумеровать элементы упорядоченной n-ки с 0 до n-1, но чётными считать a_1 (“второй”), a_3 (“четвертый”) и так далее. Так вот сложно.

Научи меня:

- Расскажите мне о массивах (списках) в Python.

Занятие 46.1. Эквивалентность

▣ Эквивалентность (Equivalence [ɪ'kwɪv(ə)l(ə)ns]) формул.

Две формулы **A** и **B** считаются \square эквивалентными тогда когда из **A** следует **B** и наоборот:

$$(A \rightarrow B) \wedge (B \rightarrow A)$$

Пишется так: $A \equiv B$ “*A эквивалентно B*”

Например: $(\neg(A \wedge B)) \equiv (\neg A \vee \neg B)$

Левую часть эквивалентности обозначим **Q**, а правую **R**:

$$Q = \neg(A \wedge B)$$

$$R = \neg A \vee \neg B$$

Построим таблицу истинности:

A	B	$A \wedge B$	Q
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

и

A	B	$\neg A$	$\neg B$	R
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

Теперь мы должны доказать, что $R \rightarrow Q$ и $Q \rightarrow R$:

Q	R	$Q \rightarrow R$	$R \rightarrow Q$	$(Q \rightarrow R) \wedge (R \rightarrow Q)$
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
0	0	1	1	1

Впрочем, здесь и без таблицы истинности видно, что столбцы **Q** и **R** одинаковы, значит $Q \equiv R$.

Формула из примера истинна в силу своей формы — независимо от содержания. Такие формулы называются логическими законами.

Вопросы для самоконтроля:

- + Что такое эквивалентность?
- + Что такое логический закон?

Занятие 46.2. Циклы

□ Циклы (loop [lu:p]) *делать...пока* и *пока...делать* (*do...while* и *while...do*).

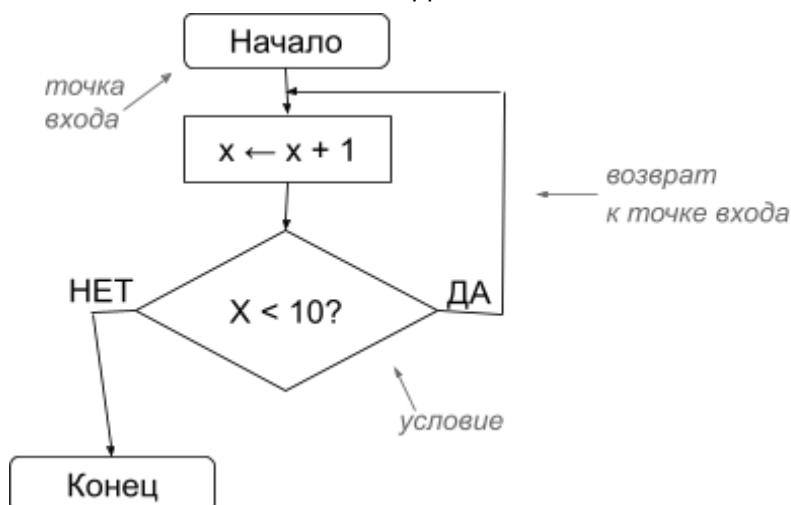
do: — точка входа в цикл

$x \leftarrow x + 1$ — тело цикла

while ($x < 10$) — условие цикла

Данный цикл будет увеличивать x на 1 пока условие цикла ($x < 10$) истинно.

Вот то же самое в виде схемы.



Теперь рассмотрим Два случая, когда до входа в цикл x было: а) 8; б) 10.

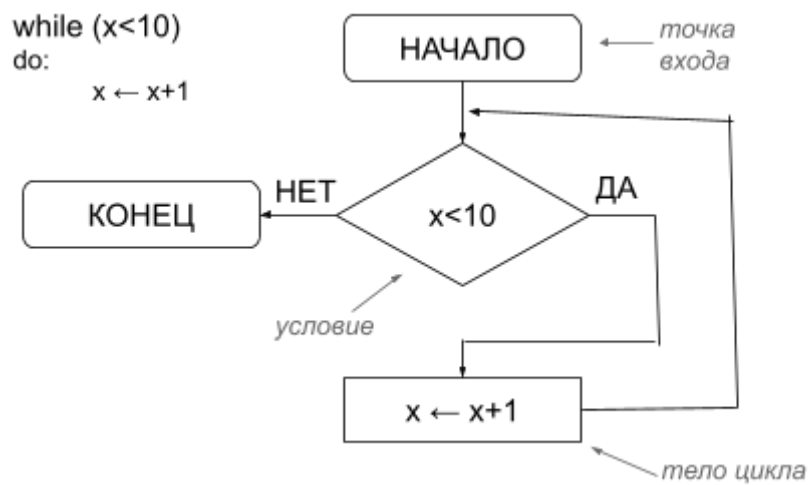
- а. Будет выполнено увеличение на 1 и x станет равным 9. Проверка $x < 10$ выдаёт Истина и прибавление единицы повторится.

Теперь $x = 10$, а $x < 10$ — Ложно.

Произойдет выход из цикла.

- б. После входа в цикл произойдёт инкремент и теперь $x = 11$. Проверка $x < 10$ выдаёт Ложь и цикл завершен.

Иначе работает цикл *while...do*. Он сначала выполняет проверку, а затем, в случае истинности условия, действие:



В случае *a* теперь:

$(x = 8) < 10 = \text{Истина}$

$x \leftarrow 8+1$

$9 < 10 = \text{Истина}$

$x \leftarrow 9+1$

$10 < 10 = \text{Ложь}$

и цикл прерван

В случае *b* теперь:

$(x = 10) < 10 = \text{Ложь}$

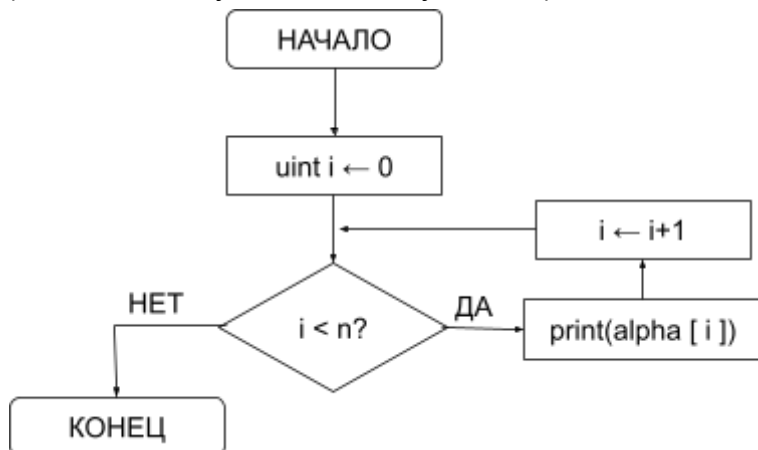
и цикл прерван, оставив *x* без изменений

Занятие 46.3. Массивы в цикле

Для перебора элементов массивов в цикле в качестве индексов обычно используются переменные с именами i, j, k , а для размеров массивов n, m, l .

Например, дан вектор длины n :

$\text{alpha} = \langle a_0, a_1, a_2, \dots, a_{n-2}, a_{n-1} \rangle$, то есть его элементы $a_i, i = 0..n-1$ (“а i тое i от нуля до эн минус один”). Выведем их при помощи *print*:



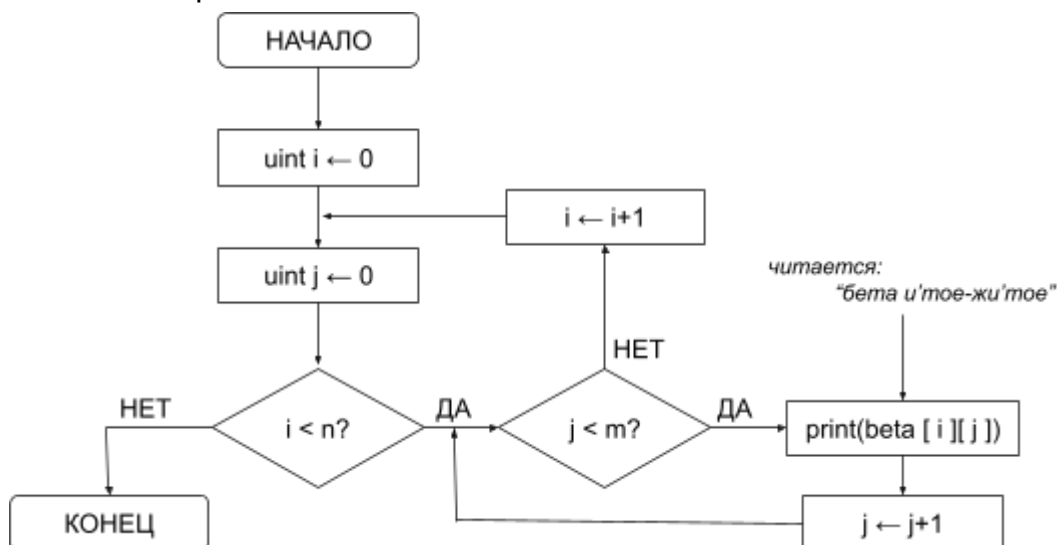
Вектор векторов называется матрицей. Например, \square матрицей (matrix) размера 3 на 2 (3x2):

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \end{bmatrix}$$

У элементов матрицы индекс из двух чисел: номера \square строки (row) и номера \square столбца (column). Фактически **A** это

$A = \langle \langle a_{00}, a_{01} \rangle, \langle a_{10}, a_{11} \rangle, \langle a_{20}, a_{21} \rangle \rangle, a_{ij} \in A, i=0,1,2, j=0,1.$

Пусть дана матрица (двумерный массив) *beta* размера $n \times m$, выведем его элементы на экран:



Трёхмерный массив — массив массивов массивов. В общем случае n -мерный вектор (массив, список) это:

`Vector<Vector<...Vector<T>>>`
 n раз Vector

Задания:

1. Напишите программу на Python, которая просит ввести 5 чисел, а затем выводит их на экран. Используйте массив и цикл while.
2. Напишите программу на Python, которая просит ввести 6 чисел. Затем каждый чётный элемент массива умножается на два, а каждый нечётный увеличивается на два. Результирующий массив вывести на экран.

Занятие 49.1. Выполнение логических операций

Порядок выполнения логических операций обычно следующий:

$\neg \quad \wedge \quad \vee \quad \rightarrow \quad \equiv$
 $\xrightarrow{\hspace{10em}}$
 раньше позже

Например:

$A \vee \neg B \wedge C \equiv D$

Так как логические операции обладают свойством ассоциативности, то подряд идущие операции можно выполнять в любом порядке:

I. $A \vee B \vee C$ II. $A \vee B \vee C$

Скобки меняют порядок выполнения. Правило такое: сначала выполняется то, что в скобках.

$(A \vee (B \rightarrow C)) \wedge D$

Задания:

1. Расставить порядок выполнения операций в формуле:

$A \vee (B \rightarrow C) \vee D \equiv P \wedge Q \wedge R \rightarrow A \vee B$

2. Узнать порядок выполнения логических операций в Python.
3. Поглядеть список логических законов. Учить не требуется, но понять каждый стоит. Можно и выучить.

4. Доказать эквивалентность в любом логическом законе при помощи таблицы истинности.

Занятие 49.2. О связи между операциями над множествами и логическими операциями.

Символ конъюнкции \wedge очень легко запомнить: если на него сесть, то можно уколоть попу и закричать “И!”. Дизъюнкция рисуется \vee — и сразу понятно, что если на неё сесть, то вся попа в такую не влезет: ИЛИ одно полупопие, ИЛИ другое.

Пересечение \cap и объединение \cup рисуются похоже, только не такие острые. В определении \cap и \cup тоже присутствуют логические И и ИЛИ:

$$A \cap B = \{ x \mid x \in A \text{ и } x \in B \}$$

“Пересечение a и b это множество всех x , таких, что x принадлежит a и x принадлежит b .”

$$A \cup B = \{ x \mid x \in A \text{ или } x \in B \}$$

“Объединение a и b это множество всех x , таких, что x принадлежит a или x принадлежит b ”.

Попробуем представить логические состояния Истина, Ложь как множества. С Ложью всё понятно, это пустое множество \emptyset . А как обозначить истину? Чтобы не выдумывать какие-то новые символы, будем считать Истину множеством, содержащим пустое множество в качестве единственного элемента $\{\emptyset\}$. Конечно так можно, почему нет? $\emptyset \neq \{\emptyset\}$.

Итак:

$$Л = \emptyset$$

$$И = \{\emptyset\}$$

Теперь любая наша логическая переменная a может быть либо $a = \emptyset$, либо $a = \{\emptyset\}$. Для двух переменных a, b множество всех возможных состояний будет $\{\emptyset, \{\emptyset\}\} \times \{\emptyset, \{\emptyset\}\} = \{\langle \emptyset, \emptyset \rangle, \langle \emptyset, \{\emptyset\} \rangle, \langle \{\emptyset\}, \emptyset \rangle, \langle \{\emptyset\}, \{\emptyset\} \rangle\}$.

Ничего удивительного.

Применим теперь к этим парам объединение:

$$\emptyset \cap \emptyset = \emptyset$$

$$\emptyset \cap \{\emptyset\} = \emptyset$$

$$\{\emptyset\} \cap \emptyset = \emptyset$$

$$\{\emptyset\} \cap \{\emptyset\} = \{\emptyset\}$$

$$\text{Имеем } S(ab) = \langle \emptyset, \emptyset, \emptyset, \{\emptyset\} \rangle = \langle 0001 \rangle = S(a \wedge b)$$

хе-хе! А что с ИЛИ?

$$\emptyset \cup \emptyset = \emptyset$$

$$\emptyset \cup \{\emptyset\} = \{\emptyset\}$$

$$\{\emptyset\} \cup \emptyset = \{\emptyset\}$$

$$\{\emptyset\} \cup \{\emptyset\} = \{\emptyset\}$$

Опять $s(a \cup b) = s(a \vee b)$!

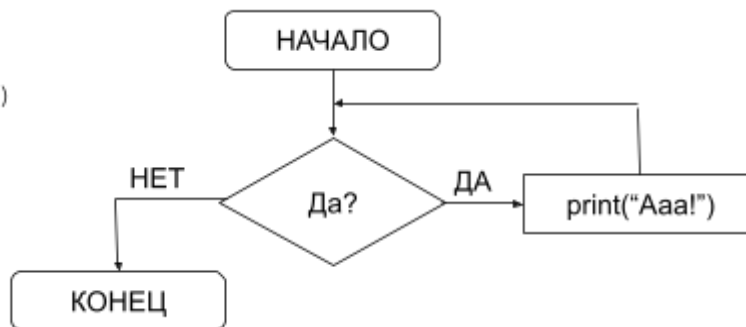
Задание:

1. Подумайте, а что первично?

Занятие 49.3. Бесконечный цикл

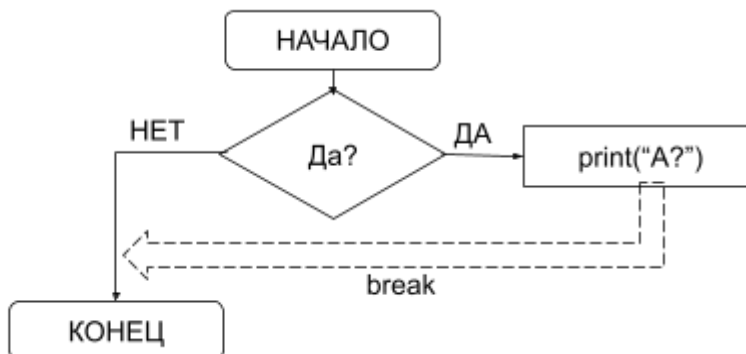
- Бесконечный цикл. Очень простая конструкция:

```
while (True)
do:
    print("Aaa!")
```



Очевидно, конец никогда не будет достигнут. Но бесконечный цикл можно прервать изнутри при помощи ключевого слова *break*:

```
while (True)
do:
    print("A?")
    break
```



Это слово работает во всех циклах. Это что-то вроде экстренной эвакуации.

Задание:

Разработайте программу, которая просит вводить слова. В случае, если пользователь вводит одно и то же слово второй раз (не подряд, а вообще), то программа пишет "You repeated!" и закрывается. Начертите её схему. Напишите её на Python, используя массив и бесконечный цикл, а также разделив программу на несколько отдельных функций.

Занятие 51.1. Операции сравнения

- Операции сравнения. Мы не будем разбирать, что значит $<$ (меньше) (хотя это интересная тема). Просто заметим, что существуют такие операции как:

- равно = (Equal)
- не равно \neq (Not equal)
- меньше $<$ (less than)

- больше или равно \geq (more or equal)
- меньше или равно \leq (less or equal)
- больше $>$ (more than)

Они нам знакомы и интуитивно понятны.

Следует чётко различать:

- операцию присвоения, которая меняет значение переменной.
- операцию равенства, которая является логической функцией установления равенства-неравенства.
- конструктор, специальный метод инициализации объекта.

Задание:

1. Даны операции \neg , \wedge , \vee , $<$. Составить из них (на бумажке) следующие функции:
 - a. \geq
 - b. $=$
 - c. \neq
 - d. $>$
 - e. \leq
2. Узнать, как выглядят операции сравнения в Python и каков порядок их выполнения с логическими операциями (\neg , \wedge , \vee).

Занятие 51.2. Двоичный код

▫ Двоичный код (binary code). Это система представления данных.

Минимальным элементом является ▫ бит (bit). Бит может иметь значение 0 или

1. Как мы уже знаем (Занятие 30. Часть 2.) 0 и 1 — это двоичные числа.

8 бит — это октет. Обычно, октет образует один ▫ байт (byte).

Пример байта: $\langle 1, 1, 1, 1, 0, 1, 0, 0 \rangle$

То есть байт — это упорядоченная восьмёрка бит. (Байт, бит — не склоняются, поэтому не битов, а бит).

В дальнейшем мы будем опускать \langle , \rangle и записывать байт так:

$\langle 0010'1111 \rangle$ или как двоичное число $b0010'1111$, разделяя для удобства чтения каждые 4 разряда. Необходимо помнить, что порядок каждого бита имеет значение, при этом массив нумеруется слева направо, числа — справа налево.

Научи меня:

- Что такое данные? Как связаны между собой понятия ▫ данных и ▫ информации?
- Килобит — это сколько бит?
- Что такое *КиБ*, *МиБ* и пр?
- Кто наживается на путанице между гига- и гигаби- ?

Занятие 52.1. Управляющие символы

uint8 — восьмибитное беззнаковое целое. Позволяет закодировать $2^8 = 256$ значений, от 0 до 255 включительно.

```
0 = b0000'0000
1 = b0000'0001
2 = b0000'0010
.
.
.
256 = b1111'1111
```

Перевод аналогичен переводу в двоичную систему счисления, а все неиспользуемые левые разряды заполняются нулями.

bool кодируется одним битом: <0> = Ложь, <1> = Истина

char — символ. В C++ это то же самое, что и `uint8`, то есть число от 0 до 255, но эти числа соответствуют различным символам. Есть множество разных кодировок символов: ASCII, Windows - 1251, KOI-8, но нас будет интересовать только первая.

Мы будем указывать одиночный символ в одинарных кавычках: 'a' — это буква a; 'A' — буква A; '.' — точка; '\n' — символ переноса строки.

Таблица ASCII устанавливает соответствия для значений `char` с 0 до 127. Первые 32 (с 0 до 31) символа называются управляющими символами — они управляют терминалом. Например таким является символ окончания строки (null terminator) '\0', код b0000'0000 или просто 0.

Стоит подчеркнуть, что приравнивание переменной

```
char ch ← 'A'
```

эквивалентно

```
char ch ← 63
```

и

```
char ch ← b0001'1111
```

Во всех этих случаях переменная `ch` будет хранить байт `<1111'1000>`. Обратите внимание, что числа мы записываем справа налево, а вектора слева направо.

Задания:

1. Почитайте об основных управляющих символах; найдите таблицу ASCII кодов.
2. Напишите программу для перевода символа в его код.
3. Напишите программу обратного перевода из десятичного числа в символ ASCII.

```
Ввод:  А
      63
Выход: 63 (0001'1111)

Ввод:   63   Выход:  А
      64           В
```

и так далее.

Занятие 52.2. Приведение типов

▫ Приведение типов (*type conversion*). Для кодирования информации в двоичном коде у нас есть только биты. Одному и тому же набору битов может соответствовать большое количество разных объектов. Например:

Хотим мы закодировать цвета. Пускай:

```
b00 = Красный   b10 = Черный
b01 = Желтый   b11 = Фиолетовый
```

Теперь, когда мы запишем `0111` будет понятно, что это чёрно-фиолетовый.

Ещё мы хотим закодировать понятия Истина и Ложь. Но двумя битами можно закодировать только 4 (2^2) объекта! Можно расширяться до 3-х бит. А что потом, когда надо будет кодировать гендерные роли? Будем кодировать логические значения (ЛЗ) так: `b00` = Ложь, `b01` = Истина. Хорошо. Но теперь, когда мы пишем `0100` — это чёрно-красный или ИстинаЛожь? Ответ: как посмотреть. Можно даже воспринимать это как чёрную Ложь или Истинную красноту. *Конкретное значение кода определяется его типом.*

```
Цвет color = b01
ЛЗ b = b01
```

Несмотря на то, что фактические значения переменных `color` и `b` одинаковы, `color` мы будем воспринимать как **чёрный**, а `b` — как **Истину**.

▫ Привести тип означает взглянуть на значение битов по-другому. Будем записывать

```
НовыйТип { переменная }
```

Например:

```
Цвет color = b00
ЛЗ boolean = ЛЗ { color }
```

Переменная `color`, значащая **красный** (<00>) приведена к новому типу и записана в переменную `boolean`, которая теперь означает **Ложь**.

Хорошо, а что если

```
Цвет color = Фиолетовый
ЛЗ b = ЛЗ{color} ?
```

Нуу... Два варианта:

1. Придётся признать значение переменной **b** недействительным (невалидным (`invalid`)), сказать, что так нельзя, обидеться и уйти.
2. Расширить определение Логического Значения:

```
b00 = Ложь           b10 = Истина
b01 = Истина        b11 = Истина
```

Задания:

1. Узнать, как приводить типы в Python.

Занятие 53. Конкатенация

Алфавитом называют непустое множество символов (значёчков). Буквами называют символы алфавита. Словом называют любую последовательность букв, в том числе длины ноль (пустое слово). Алфавиты и слова – конечные множества, то есть скажем, \mathbb{N} – не может быть алфавитом.

Если множество S – алфавит, то $A = a_0 a_1 \dots a_{n-1}$ (упорядоченная n -ка), где $a_i \in S$ – слово в алфавите S длины n . Всё это должно быть интуитивно понятно: например, в десятичной системе счисления натуральные числа – слова в алфавите $S_N = \{0, 1, \dots, 9\}$, целые числа – слова в алфавите $S_Z = S_N \cup \{-\}$. В двоичной системе счисления алфавит короток: $\{0, 1\}$. Тексты на русском тоже можно рассматривать как слова в алфавите $\{, a, \dots, Я, \dots, !, \dots\}$ и так далее и тому подобное.

Конкатенация – это приписывание слова к слову. Мы будем говорить о конкатенации применительно к спискам, то есть массивам, в том числе к строковым типам (`str` в Python или `std::string` в C++) и обозначать `..` (две точки).

```
"abc" - строка (слово в алфавите char)
'a'   - символ (буква алфавита char)
"a"   - строка из одного символа
""    - пустая строка
```

Если A, B – слова в некотором алфавите, то $A..B$ – их конкатенация. результатом которой является слово $a_0 a_1 \dots a_{n-1} b_0 b_1 \dots b_{m-1}$.

Вообще для конкатенации используются разные значки:

```
A|B
A+B
AB
A += B
```

```
A.append(B)
B.prepend(A)
A << B
```

всё это варианты записи A..B.

Вопросы:

1. Что такое алфавит? Буква? Слово?
2. Что такое конкатенация?

Задания:

1. $A="ал", k='з', E=""$, тогда $k..E..A=?$
2. Узнайте о конкатенации списков и строк в Python.
3. Строки str в Python – слова в каком алфавите? (Гуглите.)

Занятие 55.1. Интерфейсы и абстрактные классы

▣ Абстрактный метод — это фейковый, нерабочий метод, вызывать который бесполезно или даже нельзя. В Python создать абстрактный метод нам поможет ключевое слово pass (пас), которое позволяет ничего не делать:

```
def MyAbstractFunction: arg1, arg2
    pass
```

Благодаря *pass* эта функция ничего не сделает, аргументы не будут использованы, функция ничего не вернёт.

Задание: почитать о ключевом слове *pass*.

Абстрактный класс — это класс, все методы которого абстрактны. Зачем вообще нужен класс с нерабочими методами? Зачем вообще могут понадобиться методы, которые нельзя вызывать? Ответ: чтобы разработчики не вызывали эти методы. И тут мы переходим к понятию интерфейса.

□Интерфейс — это некоторая договоренность, контракт. Договоренность о способах взаимодействия. Например, вы говорите с подружкой об условном знаке: "Я постучу вот так **тук, тук** и ты поймёшь, что это свои". Подруга может рассказать об этом вашему другу, тогда он тоже сможет зайти. В этом примере два объекта (подруга и друг) используют один и тот же интерфейс для доступа к третьему объекту (вам). А вы *имплементируете* (проводите в жизнь, реализуете) интерфейс, то есть обеспечиваете его работу (слушаете стуки и открываете дверь на **тук, тук**).

Сам по себе интерфейс — вещь бесплотная. Какие-то классы должны его реализовывать, что бы он обрёл конкретный смысл. Договоренность о стуках не имеет смысла, если никто не слушает дверь. Реализовывать интерфейс может множество классов; например вы можете передоверить слушать дверь собаке.

Далеко не во всех языках программирования есть понятие интерфейса. Тогда используются абстрактные классы, то есть классы, все методы которых абстрактны. В Python'e нет абстрактных методов (нет ведь?), поэтому мы будем их имитировать при помощи *pass*.

"Имплементировать / реализовывать интерфейс" в случае с абстрактными классами будет означать "унаследоваться и имплементировать все его методы".

```
class ACar:
    get_brand: void → String
    get_max_speed: void → uint
```

Здесь **A** в названии означает **abstract**.

Очень распространено использование буквы I (interface) для указания, что это именно интерфейс: *ICar*, что читается [ai'ka:] (Я — автомобиль). Имплементации интерфейса отличается от простого наследования именно тем, что интерфейс не навязывает того как имплементировать методы. При классическом наследовании класс-родитель должен быть рабочим, содержать внутреннюю.

Реализовывать этот интерфейс можно множеством способов. Но сначала посмотрим, как им воспользоваться:

```
class CarPrinter:
    print: ACar car → void
        print("This is ")
        print(car.get_brand())
        print(" with max speed ")
        print(car.get_max_speed())
        print(".\n")
```

Этот класс умеет печатать информацию о любом классе, реализующем интерфейс *ACar*. Теперь:

```
class DodgeViper: parent ACar
    get_brand: void → String r
        r ← "Dodge"
    get_max_speed: void → uint r
        r ← 300
```

А можно вот так:

```
class CarManual: parent ACar
    field String brand
    field uint max_speed
    constructor: String brand, uint max_speed
        self.brand ← brand
        self.max_speed ← max_speed
    get_brand: void → String r
        r ← self.brand
    get_max_speed: void → uint r
        r ← self.max_speed
```

Классы DodgeViper и CarManual не имеют между собой ничего общего кроме интерфейса, все их внутреннее устройство отлично. Но оба представляют собой воплощение в жизнь абстрактного описания ACar. Попробуем что получилось:

```
CarPrinter p{}
DodgeViper dv{}
p.print(dv)
CarManual m{"Mazda", 200}
p.print(m)
```

Заметим, что

```
ACar a{}
p.print(a)
```

приведёт к ошибке, нельзя вызывать абстрактный метод.

В этом примере класс ACar — базовый, но из абстрактных классов можно строить более сложные интерфейсы:

```
class ANamed:
    get_name: void → String
class AMovable:
    move_to: Place → void
class ANamedMovableCar: parents ACar, ANamed, AMovable
```

Вопросы для самоконтроля:

- что такое интерфейс?
- что такое абстрактный класс, метод?

Задание:

Реализовать на Python примеры из занятия.

Факультатив:

Вместо *pass* можно использовать

```
throw "This method is an abstract one"
```

Занятие 55.2. Операционная система и разрядность CPU

Научи меня:

1. Расскажите мне об операционных системах (ОС). Какие бывают? Зачем нужны? Чем отличаются? Приведите пример.
2. Что такое центральный процессор? Зачем нужен? Из чего состоит? А, лол, или это было уже про RISC и CISC? Тогда: что такое разрядность CPU? На что влияет? Какая бывает? Что такое машинное слово?
3. Что значит Little Endian / Big Endian? О чем это? Какой у вас процессор: LE или BE?

Занятие 56. Пишем 'хо'

Теперь мы готовы разобраться с проектом **хо**. Первое, что мы решим, будет то, как нам обозначать клетки игрового поля: двумя числами (строка, столбец)? Или одной? Так как клеток всего 9, то тут проще обойтись одним числом. Плюс, это тот формат, в котором будет указывать свой ход игрок (согласно ТЗ). Будем обозначать клетки числом от 0 до 8 слева направо, сверху вниз. Тогда поле можно представить себе, как упорядоченную 9-ку, массив клеток. Второе: как обозначать крестик, нолик и пустую клетку? Enum-ом. Я не заставлю вас использовать библиотеку Enum в этот раз, но вы можете сделать это именно таким способом, импортировав Enum в этот же модуль. Но я назначу три числовые константы в виде полей модели игрового поля. Конкретные числа не важны, важно лишь то, что они должны быть разными. С учётом всего сказанного интерфейс игрового поля будет иметь следующий вид:

```
class IXOField:
    field uint CELL_EMPTY = 0
    field uint CELL_X = 1
    field uint CELL_O = 2
    get: void → Array <uint, 9> field
    set: uint cell, uint value → bool success
    field uint WINNER_NONE = 0
    field uint WINNER_X = 1
    field uint WINNER_O = 2
    field uint WINNER_DRAW = 3
    get_winner: void → uint winner
```


Здесь *get_winner* позволит нам спрашивать победителя у модели, все возможные значения функции представлены соответствующими константами. Если игра ещё идёт, то победителя нет (*WINNER_NONE*).

В Python придется инициализировать конкретные поля в конструкторе:

```
class IXOField(Object)
    def __init__:
        self.CELL_EMPTY = 0
        self.CELL_X = 1
        . . .
```

Итак, с основной моделью проекта все понятно. Дальше проще всего описать вьюху. Так как никакого преобразования данных в контроллере не требуется — игровое поле будет просто выводиться на экран:

```
class IXOView:
    print_field: Array<uint, 9> field → void
    print_winner: uint winner → void
```

Обращу ваше внимание на метод *print_winner*. Он принимает *uint*, фактически он будет принимать то, что возвращает метод *IXOField.get_winner*, точнее 3 из 4-х констант *WINNER_X(O,DRAW)*. С перечислением *Winner* было бы яснее.

С моделью игрока тоже все просто: нам нужен способ спрашивать у игрока его ход, назовем это *ask*. Но ещё я знаю будущее, поэтому мы наделим игрока способностью сообщать за какую сторону он играет. С этим связан вопрос: что́ будет решать за какую сторону игрок? Если выбор будет происходить вне контроллера, то устанавливать сторону можно в конструкторе модели игрока. Мы возложим распределение сторон на контроллер, поэтому нам придется добавить метод *set_side* (*сеттер*) и соответствующий *getter*. Посмотрите слова *set* и *get* в словаре.

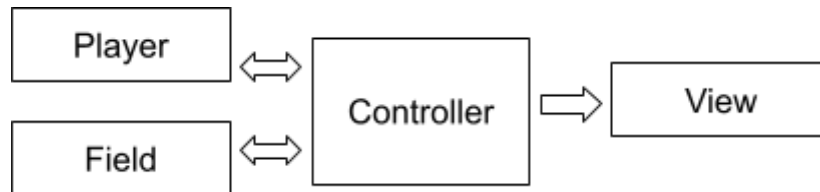
```
class IXOPlayer:
    ask: Array<uint,9> field → uint cell
    fiels uint SIDE_NONE = 0
    fiels uint SIDE_X = 1
    fiels uint SIDE_0 = 2
    set_side: uint side → void
    get_side: void → uint side
```

Теперь займёмся описанием контроллера. В конструктор мы будем передавать модель игрового поля, двух игроков и вью. Что ещё надо? Ну, метод, который будет начинать (и заключить в себе) весь игровой процесс. Назовем его просто *play*:

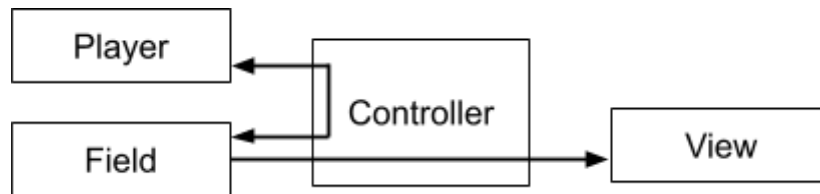
```
class IXOController:
    constructor:    IXOField, IXOPlayer,
                   IXOPlayer, IXOView
    play: void → void
```

Вот и все. Замечу ещё, что двух игроков можно передавать не по-отдельности, а массивом. У такого способа есть плюсы, но об этом потом.

Итак, мы имеем следующую схему:



Здесь стрелочками указано перемещение информации. Фактически данные будут ходить вот так:



Задача контроллера в нашем случае не велика. Самым сложным модулем проекта является игровое поле: именно он будет следить за правильностью ходов и определять победителя.

Задание:

Создайте для проекта **хо** директорию, в ней создайте 5 файлов `хо.ру`, `ixofield.ру` и так далее.

В `хо.ру` создайте функцию `__main__`, а в четырёх других опишите абстрактные классы `IXOField` и так далее.

Занятие 58. XOPlayer

Начнём реализовывать **хо** с модели игрока. В дальнейшем вы убедитесь, что в хорошо спроектированном проекте можно реализовывать любую часть отдельно — и начать, соответственно, можно с любой.

Нам надо унаследоваться от абстрактного `IXOPlayer` и имплементировать три метода: `ask`, `set_side` и `get_side`. Два последних очень просты, это сеттер и геттер, для которых нам нужно завести поле, которое они будут устанавливать (`set`) и отдавать (`get`). Мы назовём это поле `_side`. Почему с “_”? Потому что мы хотим обозначить, что это поле — для *внутреннего* использования — *внутри* класса. Такое поле называется приватным полем, а геттер и сеттер обеспечивают доступ к нему. Использование сеттеров и геттеров, то есть методов класса, для доступа к полям класса позволяет осуществлять этот доступ по сложным правилам, менять эти правила легко и безболезненно для других частей программы. Хочешь изменить поле? Используй сеттер. Хочешь получить поле? Используй геттер. Такие методы скрывают реализацию доступа к данным внутри класса, скрывают от внешнего (по отношению к классу) мира. Скрытие реализации, то есть того как именно устроено, от внешнего мира называется инкапсуляцией. В Python нет понятия приватного поля, поэтому мы просто добавим “_” — это распространённая практика.

Поле `_side` должно быть инициализировано при создании экземпляра класса значением `SIDE_NONE` (`Side.None` для Enum), чтобы свежесозданные экземпляры могли рапортовать что ни к крестикам, ни к ноликам они пока ещё не принадлежат.

Итак, создаём файл `xoplayer.py`, импортируем `IXOPlayer` и создаём класс

```
class XOPlayer: parent IXOPlayer
    private field uint _side
    constructor:
        self._side ← SIDE_NONE
    set_side: uint side → void
        self._side ← side
    get_side: void → uint side
        side ← self._side
```

Вот и всё. Может возникнуть вопрос: а зачем это всё? Пока совсем не нужно; но я знаю будущее.

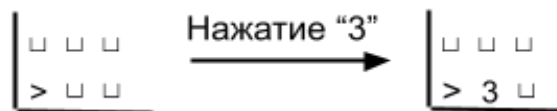
Вопросы для самоконтроля:

1. Что такое приватное поле?
2. Что такое инкапсуляция и зачем нужны геттеры-сеттеры?

Задание:

Реализовать в классе `XOPlayer` метод `ask: Array<uint,9> field → uint cell`. Он принимает игровое поле, а возвращает выбор игрока куда ставить его значёчек. А на кой чёрт нам получать игровое поле, если игрок и так видит его на экране? Проверять, не походил ли он на уже занятую клетку? Нет, этим занимается модель игрового поля. Значит это бесполезный аргумент метода — **неиспользуемый аргумент**, то есть такой аргумент, который не будет использован при реализации метода / функции. В C++ принято лишать такие аргументы имени, но в Python имя `field` придётся оставить.

Метод `ask` должен вывести на экран символ “>” (как об этом сказано в Техническом задании!), затем нужно считать число с клавиатуры (от 1 до 9) (из стандартного ввода), затем перевести его в формат обозначения ячеек внутри программы (от 0 до 8) и вернуть его из метода.



Техническое задание ничего не говорит о том, что делать в случае, если пользователь ввёл неподходящее число (или вообще абракадабру). Воспримем это как инструкцию и не будем ничего делать, а просто переспросим пользователя снова выведя “>” (с новой строки).

Если не сразу понятно как это сделать на Python — начните с рисования схемы алгоритма.

Желаю удачи!

Занятие 59.1. Foreach

Foreach — это цикл, пробегающий по элементам некоторого множества. Пусть

$L = \langle a, b, c \rangle$

тогда

```
foreach item from L
do:
    fun(item)
```

выполнит последовательно

```
fun(a)
fun(b)
fun(c)
```

в той последовательности, в которой элементы расположены в упорядоченном множестве.

Задание:

Узнайте о `foreach` в Python. Напишите функцию, которая будет выводить элементы переданного списка через пробел.

Занятие 59.2. Foreach на неупорядоченном множестве

В общем случае абсолютно не обязательно, чтобы множество для пробегания было упорядоченным. Представим *foreach* как функцию от двух аргументов: множества для пробегания и функцию — тело цикла. Пусть функция *get_item* от множества возвращает какой-то один элемент этого множества, пока элементы не закончатся — в этот момент пусть *get_item* вернёт, скажем, \emptyset

Тогда

```
foreach: S, body → void
var item ← get_item(S)
while(item ≠ ∅)
do:
    body(item)
    item ← get_item(S)
```

Таким образом задача сводится к построению функции *get_item*.

Занятие 60. XOView

В Техническом задании не указано, что выводить в случае ничьей. Будем выводить строку “Draw”.

Замечу ещё раз, что *print_winner* принимает 3 из 4 константы *IXOModel.WINNER_**, поэтому, для доступа к этим константам при имплементации выю надо будет импортировать модуль *ixofield.py*.

А почему просто не написать “если 1, тогда победитель X”?

Числа, встречающиеся в коде программ без пояснений называют ▣магическими числами (у этого словосочетания есть и другие значения), потому что это всё выглядит непонятно для непосвященных, как магия. Например:

```
1 | if (height( )+ 3 > MAX_HEIGHT)
2 | do:
3 |   reset(15, 3)
```

Что за 3? Что за 15? Откуда они взяты? Что этот код означает? Для того, чтобы это понять, требуется некое тайное знание — которое есть только в голове того, кто это написал (и то он сам уже забыл).

А вот тройка, тройка в строке 1 и строке 3 — это одна и та же тройка? Ну, по смыслу. Да? Нет? Как понять? Хорошо, допустим одна и та же. Теперь надо изменить эту тройку на, скажем, пятёрку. А между строками 1 и 3 появилось 15 новых строк кода и это скромное *reset (15, 3)* на их фоне не заметно. Даже у автора этого кода очень большие шансы поменять в строке 1 и забыть поменять в строке 3. А затем удивляться почему не работает.

Потратив 4,7 дополнительные секунды можно сделать красивше:

```
const uint border = 3
if (height( )+ border > MAX_HEIGHT)
do:
    reset(BOX_SIZE, border)
```

Не позволим магическим числам пробраться в наш проект и вместо 1 будем использовать `IXOField.WINNER_X`.

Вопросы:

1. Что такое магические числа и чем они чреватые?

Задание:

Написать вью `XOView`.

Занятие 62. XOField #1

Теперь мы приступим к самой сложной части проекта `xo` — модели игрового поля. Что она должна делать?

1. Хранить игровое поле. В виде массива (списка) положительных чисел длины 9.
2. Проверять, что игрок ходит на пустую клетку.
3. Определять победителя.

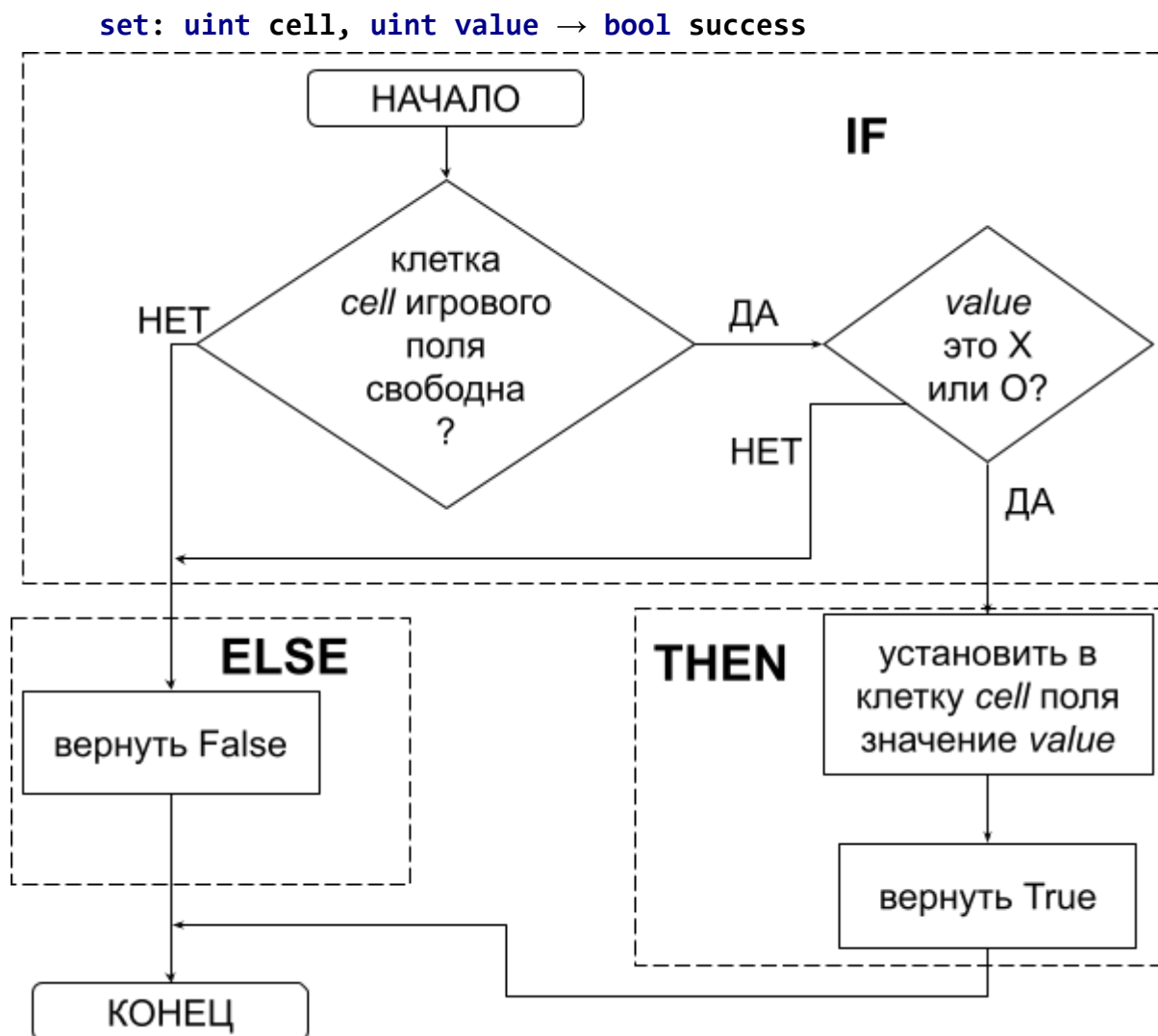
Нам необходимо имплементировать три метода интерфейса `IXOField`. Нарисуем блок-схемы алгоритмов для каждого из них.

`get: void → Array<uint, 9> game_field`



Довольно просто. Надо будет завести приватное поле `_game_field`, которое инициализировать значениями `CELL_NONE` (почему?) в конструкторе.

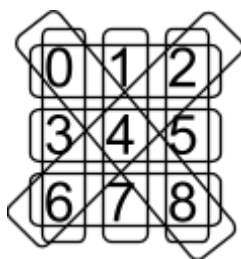
Метод `get` — простой геттер.



Как видно из схемы, метод `set` должен позволить вставить в клетки только `Cell.X` и `Cell.O`, но не `Cell.None`.

`get_winner: void → uint winner`

На данном этапе мы будем проверять победителя “в лоб” — полным перебором. Разделим игровое поле на тройки которые ведут к победе:



три горизонтальных,
три вертикальных
и две диагональные.
Итого 8.

Поле — это упорядоченное множество клеток. Каждая клетка имеет номер и значение. Обозначим множество значений как $V = \{\text{None}, X, O\}$ и множество номеров как

$N = \{0, 1, \dots, 8\}$. Тогда поле то $F = \langle c_0, c_1, \dots, c_8 \rangle$, где $c_i \in V, i \in N$.

$$F = \langle c_0, c_1, c_2, \\ c_3, c_4, c_5, \\ c_6, c_7, c_8 \rangle$$

Рассмотрим интересующие нас вопросы:

1. Закончена ли игра? (W)
2. Победитель — крестики? (W_x)
3. Победитель — нолики? (W_o)
4. Ничья? (D)

Как видно, это вопросы, на которые можно ответить только Да (True) или Нет (False). Такой формат позволит нам работать с этими вопросами как логическими выражениями W, W_x, W_o и D . Утвердительный ответ на один из трёх последних вопросов исключает два других:

$W_x \rightarrow \neg W_o \wedge \neg D$ “Если победитель — крестики, то нолики не победитель и в игре не ничья”.

$$W_o \rightarrow \neg W_x \wedge \neg D$$

$$D \rightarrow \neg W_x \wedge \neg W_o$$

Также очевидно, что ответ на первый вопрос W можно составить из трёх оставшихся: “Игра закончена тогда и только тогда, когда победитель крестики, нолики или ничья”:

$$W \equiv W_x \vee W_o \vee D$$

Теперь о критериях победы: если в одной из восьми троек (любой) значения совпадают и равны X или O , то вот он победитель. Записать проверку одной тройки (возьмём для примера диагональ $\{c_2, c_4, c_6\}$ (почему тройка это множество, а не упорядоченное множество?)) можно так:

$$(c_2=c_4 \wedge c_4=c_6 \wedge c_6=X \rightarrow W_x) \vee (c_2=c_4 \wedge c_4=c_6 \wedge c_6=O \rightarrow W_o)$$

Длинно. Попробуем упростить. Представим W_x как $winner = X$ и W_o как $winner = O$:

$$(c_2=c_4 \wedge c_4=c_6 \wedge c_6=X \rightarrow winner=X) \vee (c_2=c_4 \wedge c_4=c_6 \wedge c_6=O \rightarrow winner=O)$$

Кажется, что это можно упростить до $c2=c4 \wedge c4=c6 \rightarrow \text{winner}=c6$,
но это не так: надо ещё убедиться, что тройка заполнена X или O, но не *None*.
Поэтому мы добавим проверку

$$c2 = c4 \wedge c4 = c6 \wedge (c6 = X \vee c6 = O) \rightarrow \text{winner} = c6$$

которую можно записать проще:

$$c2 = c4 \wedge c4 = c6 \wedge c6 \neq \text{None} \rightarrow \text{winner} = c6$$

Отлично! Осталось повторить 8 раз

$$c0 = c1 \wedge c1 = c2 \wedge c2 \neq \text{None} \rightarrow \text{winner} = c2$$

$$c3 = c4 \wedge c4 = c5 \wedge c5 \neq \text{None} \rightarrow \text{winner} = c5$$

. . .

$$c2 = c4 \wedge c4 = c6 \wedge c6 \neq \text{None} \rightarrow \text{winner} = c6$$

Задания:

1. Почитать об *elif* в Python.
2. Написать конструктор, *get* и *set* класса *XOField*. Метод *get_winner* пока не трогать.

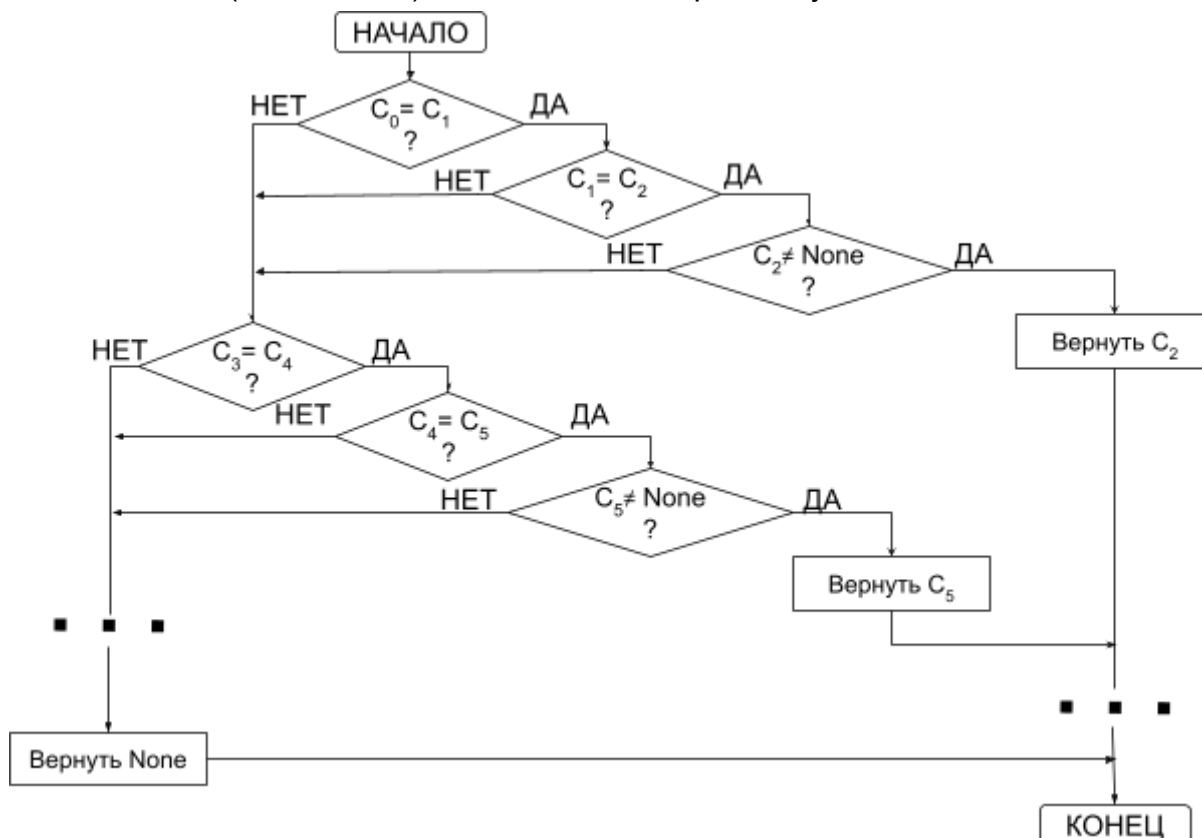
Занятие 63. RAM

Научи меня:

1. Что такое RAM? Какую роль RAM выполняет в компьютере?
2. Что такое кэши процессора?
3. Что такое SDRAM и SSRAM? Где и почему применяется? ← факультатив.

Занятие 64. XOField#2

Чтобы получить что-то практическое из полученных 8-ми формул, нам достаточно заменить логическое “то победитель c_i ” ($\rightarrow \text{winner} = c_i$) на “вернуть победителя c_i ” ($\text{winner} \leftarrow c_i$). Тогда схема алгоритма будет иметь вид:



Блок каждой тройки клеток будем описывать оператором *if/elif...then*:

```

if (c0 = c1 ∧ c1 = c2 ∧ c2 ≠ CELL_NONE)
then:
  if(c2 = CELL_X)
  then:
    winner ← WINNER_X
  else:
    winner ← WINNER_O
elif (c3 = c4 ∧ c4 = c5 ∧ c5 ≠ CELL_NONE)
then:
  . . .
else:
  winner ← WINNER_NONE

```

Несмотря на то, что фактические значения констант CELL_X(O) и WINNER_X(O) совпадают, мы вынуждены делать “перевод” из CELL_* в WINNER_* — на случай, если их значения когда-нибудь перестанут совпадать. Необходимости в переводе можно было избежать на этапе проектирования интерфейса унифицировав значения клеток, победителей и сторон игры, уложив всё в понятие, например SideAndWinner = {None, X, O, Draw}.

Задание:

Имплементируйте метод определения победителя в XOField, помолвившись Св. Копи Пэйсту. Подумайте: как можно улучшить этот код?

Занятие 65. Цикл For (для)

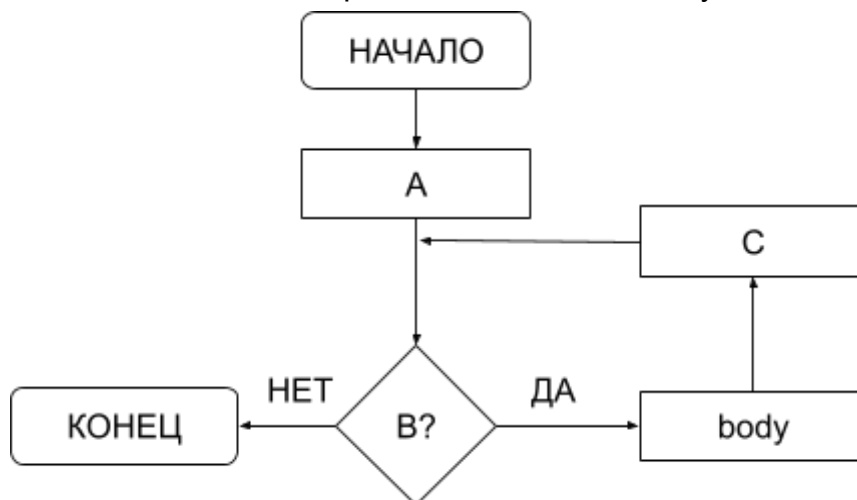
Рассмотрим ещё один тип цикла \square —for. Он имеет вид

```

for(A;B;C)
do:
  body

```

где A — место для инициализации переменных-индексов, B — для проверки условия, C — для модификации индексов и body — тело цикла.



Как видно, его легко заменить циклом while:

```

A
while(B)
do:
    body
  C
  
```

Самый популярный способ использования цикла for — это перечисление натуральных чисел от 0 до N-1:

```

for (uint i = 5; i < size(A); i ← i+2)
do:
    print(A[i])
  
```

выведет только чётные элементы из A начиная с A[5]

Обычно язык не запрещает модифицировать переменную в теле цикла, как и создавать в другом месте

```

int i = N-1
for(; i ≥ 0;)
do:
    print(i)
    i ← i-1
  
```

Вопрос: Что выведет этот цикл? Почему *int*?

```

for(;True;)
do:
    print("infinite loop")
  
```

Ограничиваться одной переменной тоже никто не заставляет.

```
for(uint i = 2, uint j = 64; i ≠ j; i ← i·i; j ← j/2)
do:
    print (i : ' ' : j)
```

Вопрос: Что должен вывести этот цикл?

В Python такого цикла нет.

Занятие 66. XOController

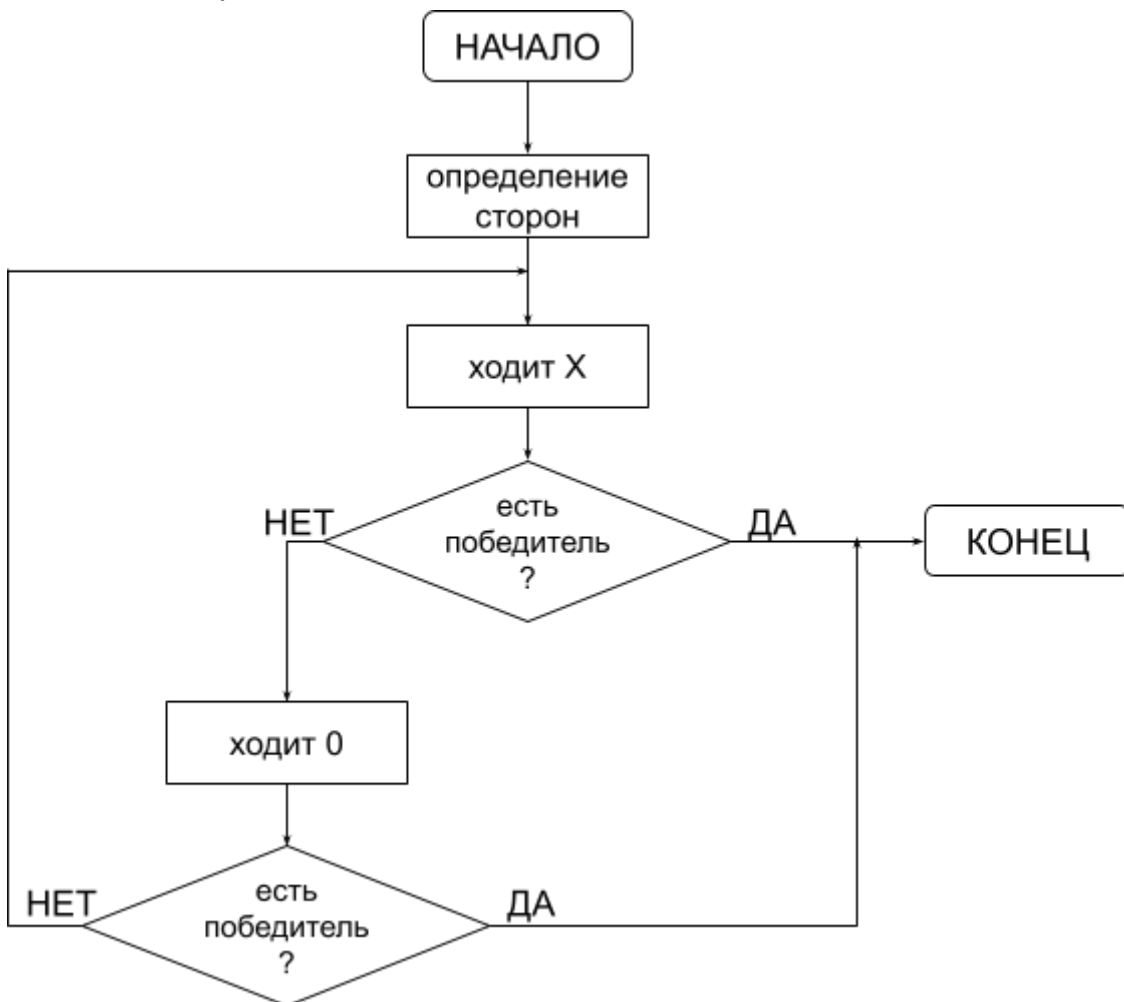
Как уже отмечалось, контроллер в **xo** вещь несложная. Приступим к реализации.

Очевидно, что принятые в конструкторе / согласно интерфейсу (IXOController) модели и вью надо хранить. Заведём для них приватные поля:

```
class XOController: parent IXOController
    private field IXOField _f
    private field IXOPlayer _p0
    private field IXOPlayer _p1
    private field IXOView _v
```

и будем инициализировать их в конструкторе (`self._f = field` и так далее).

Для имплементации метода `play` разберёмся, что такое крестики-нолики. Первым делом игроки распределяют роли (стороны) — кто за X, кто за O. Затем игроки поочерёдно ходят (сначала X, потом O, потом снова X и так далее), после каждого хода проверяя, нет ли победителя. Таким образом схема игры может быть изображена так:



Отлично! Остаётся переписать это в терминах наших интерфейсов.

“Определить стороны” означает установить сторону каждому игроку (`set_side`)

“Ходит X” означает, что на экране выводится игровое поле (`print_field`), после чего игрок X выбирает клетку (`ask`). Далее, если клетка не занята, то в неё ставится X (`set`). Аналогично для O.

“Есть победитель” означает спросить победителя у модели игрового поля (`get_winner`) и если ответ не `WINNER_NONE`, то игра закончена.

Задание А.

Изобразить подробную схему алгоритма метода `play` в терминах наших интерфейсов.

Задание В.

Написать класс XOController, имплементировать метод play согласно составленной вами схеме.

(Подсказки: вам может помочь бесконечный цикл и цикл do...while)

(Возможное решение на псевдокоде:

```

play: void → void
  var x ← self._p0
  var o ← self._p1
  x.set_side(IXOPlayer.SIDE_X)
  o.set_side(IXOPlayer.SIDE_O)
  while (True)
  do:
    _v.print_field(_f.get())
    do:
      uint x_move ← x.ask(_f.get())
      bool x_success ← _f.set(x_move, x.get_side())
    while (¬x_success)

    var winner ← _f.get_winner()
    if (winner ≠ IXOField.WINNER_NONE)
    then:
      break

    _v.print_field(_f.get())
    do:
      uint o_move ← o.ask(_f.get())
      bool o_success ← _f.set(o_move, o.get_side())
    while (¬o_success)

    winner ← _f.get_winner()
    if (winner ≠ IXOField.WINNER_NONE)
    then:
      break

```

В этом примере *self*. при обращении к полям опущены)

Вопросы:

1. На какие блоки можно поделить схему метода `play`?
2. Какие вы видите закономерности в приведённом мною псевдо коде метода `play`? Как его можно улучшить?

Занятие 67. Регистры; шина

Научи меня:

1. Что такое регистры процессора?
2. Что такое компьютерная шина и зачем она нужна? Приведите примеры.

Занятие 68. хо.ру

Если не считать ничьих, то наша программа почти готова. Теперь в `__main__`:

```
var p0 = XOPlayer{}
var p1 = XOPlayer{}
var v = XOView{}
var f = XOField{}
var c = XOController{f, p0, p1, v}
c.play()
```

Вуаля!

Задание:

Убедитесь, что всё работает, исправьте ошибки, если обнаружатся.

Занятие 69. Рефакторинг

Отлично, у нас есть рабочая программа! Но мне не нравится моё решение задания В из 66. Во-первых, выглядит слишком длинно для такого простого метода – ~25 строк. Во-вторых, внутри бесконечного цикла два явно похожих куска! Они отличаются только тем, что первый относится к X, а второй к O. Будем это исправлять.

Изменение кода программы не меняющие её функционал называется рефакторинг (refactoring). Рефакторинг может проводиться по разным причинам. (В русском допустимо ударение и на второй, и на третий слоги.)

Разделим метод `play` на части согласно схеме алгоритма (66):

1. Определение сторон
2. Ходит X
3. Проверка победителя
4. Ходит O

Для 1 заведём приватный метод `_set_sides`.

```
private _set_sides: void → void
```

```

self._x ← self._p0
self._o ← self._p1

```

Да, понадобятся ещё поля `_x` и `_o`. (Есть и другие варианты.) Можно вообще отказаться от полей `_p0` и `_p1`, всё это выглядит лишним.

2 и 4: Ходит X отличается от Ходит O только игроком! Заведём приватный метод для хода `_move(_turn?)`:

```

private _move: IXOPlayer player → void
_v.print_field(_f.get())
do:
    uint move ← player.ask(_f.get())
    bool success ← _f.set(move, player.get_side())
while(¬success)

```

3: Нам надо понимать, идёт ли ещё игра или уже нет? Ответ на этот вопрос нам будет давать метод `_is_playing`.

```

private _is_playing: void → bool r
r ← _f.get_winner() = IXOField.WINNER_NONE

```

Есть и второй вариант: завести поле и метод для его обновления.

```

private field bool _is_playing
private update_is_playing: void → void

```

Но в данном случае это лишнее.

Вот и всё: распахав всё по методам, мы получаем вот такую красоту:

```

play: void → void
_set_sides() // процедура!
do:
    _move(self._x)
    if(¬_is_playing())
    then:
        break
    _move(self._o)
while (_is_playing())
_v.print_winner(_f.get_winner())

```

Лаконично и почти в 3 раза короче. Без бесконечного цикла. А самое главное: понять этот код проще.

Вопросы:

1. Что такое рефакторинг?
2. Что есть самое главное?

Задание:

Проведите рефакторинг метода `play` в своём проекте.

Занятие 70. XOField #3

Осталось разобраться с ничьёй.

Взглянем на метод `get_winner`. Если вы сделали так, как я просил — используя копи-пэйст, — то метод представляет собой стену кода. Что-то исправить или изменить здесь не так-то просто!

Задание

Провести рефакторинг метода `get_winner`.

Решение: например, завести методы

```
private _is_filled: uint i, uint j, uint k → bool r
    r ← ci = cj ∧ cj = ck ∧ ci ≠ CELL_NONE

private _cell_to_winner: uint i → uint r
    if (ci = CELL_X)
    then:
        r ← WINNER_X
    else:
        r ← WINNER_O
```

Итогом этого всё равно будет куча *if/elif*.

Для того, чтобы упростить сильнее, надо понять при каком условии *меняется* переменная `winner`. Тогда, когда у нас *появился* победитель. А если победитель уже есть, то ничего проверять уже не надо. Соответственно, проверять тройки надо только тогда, когда `winner=None`.

```
private _check_line: uint i, j, k, winner → uint r
    if (winner = WINNER_NONE
        ∧ _is_filled(i, j, k))
    then:
        r ← _cell_to_winner(i)
    else:
        r ← winner
```

Тогда метод `get_winner` приобретает следующий вид:

```
get_winner: void → uint winner
    winner ← WINNER_NONE
    winner ← _check_line(0, 1, 2, winner)
    winner ← _check_line(3, 4, 5, winner)
    . . .
    winner ← _check_line(2, 4, 6, winner)
```

9 строк — уже лучше, не правда ли?

Занятие 71.1. Адресация RAM

Задание:

Узнайте об адресации RAM и адресах памяти.

Занятие 71.2. Итераторы

Мы уже говорили о *foreach* и некоторой функции *get_item*, позволяющей пробегать множества — упорядоченные или нет.

Попробуем написать класс, который будет решать эту задачу для упорядоченного множества.

```
class AIterator:
    constructor: Vector<T>
    get: void → T item
    next: void → void
    is_end: void → bool
```

Это интерфейс для итератора. При создании экземпляра ему передаётся массив (динамической длины) с элементами типа T. Методы:

get — для получения элемента множества;

next — для перехода к следующему элементу;

is_end — для обозначения достижения конца массива.

Задание:

Напишите Alterator на Python. Напишите класс Iterator, имплементирующий этот интерфейс. Убедитесь в правильности работы на такой функции:

```
def print_list(list):
    i = Iterator(list)
    while not i.is_end():
        print(i.get())
        i.next()
```

Занятие 72. XOField #4

Нужно провести некоторый анализ игры, чтобы понять, что такое ничья.

Задание.

Прежде, чем идти дальше разберитесь — что же такое ничья.

Насколько я могу понять, ничья — это когда остались две пустые клетки, но никто не победил. Если обозначить через E количество пустых клеток, то ничья это

$$D \equiv \neg W_x \wedge \neg W_o \wedge E=2.$$

Отлично, значит нам надо считать пустые клетки! Добавим метод

```
private _empty_cells_count: void → uint count
  count ← 0
  foreach cell from get()
do:
  if (cell = CELL_NONE)
do:
  count ← count +1
```

Нет.

Зачем нам бегать циклом по массиву, если можно считать клетки сразу, когда они заполняются — в методе *set*?

Две пустые клетки это $9 - 2 = 7$ заполненных, заполненная клетка — 1 ход. Будем считать ходы.

Заведём для этого

```
private field uint _moves_count
```

Задание.

Доделать XOField до конца, то есть добавить определение ничьей.

Занятие 74. XOField #5

Ура, всё сделано согласно Техническому заданию хо 0.1!

Да, но у меня всё ещё есть претензии к реализации проекта. И не нравится мне метод *get_winner*: каждый раз 8 (восемь!) проверок... Многовато. Не хочу полный перебор, хочу умнее!

Задание.

Родите мне умный способ определения победителя классом XOField.

Условия:

1. Можно модифицировать любую часть класса (кроме родительского интерфейса IXOField).
2. Количество проверяемых “троек” не должно превышать четырёх.

Решение

От чего зависит победа? От ходов! От них и будем плясать.

Во-первых, победа невозможна, пока на поле не появятся три одинаковых значка. Крестики ставят третий X на пятом ходу игры, а значит первые четыре хода вообще ничего не надо проверять победителя просто не может быть.

Во-вторых, победа может случиться только в момент хода — и ни в какой другой. поэтому выявлять победителя мы будем в методе *set* после каждого успешного хода.

Наши восемь “троек” это три строки:

$$R_0 = \{0, 1, 2\}, \quad R_1 = \{3, 4, 5\}, \quad R_2 = \{6, 7, 8\}$$

три столбца:

$$C_0 = \{0, 1, 2\}, \quad C_1 = \{3, 4, 5\}, \quad C_2 = \{6, 7, 8\}$$

и две диагонали:

$$D_{08} = \{0, 4, 8\} \quad D_{62} = \{2, 4, 6\}$$

Самый простой из умных вариантов — это [□]Захардкодить (to hardcode) соответствие между клеткой, на которую ходит игрок, и набором “троек” для проверки. Выпишем эти соответствия в виде таблицы

Ход игрока	Строка	Столбец	Диагональ
------------	--------	---------	-----------

0	R_0	C_0	D_{08}
1	R_0	C_1	
2	R_0	C_2	D_{62}
3	R_1	C_0	
4	R_1	C_1	D_{08}, D_{62}
5	R_1	C_2	
6	R_2	C_0	D_{62}
7	R_2	C_1	
8	R_2	C_2	D_{08}

Каждому ходу соответствует от двух до четырёх “троек” для проверки.

Реализовать эту табличку в коде нам поможет трёхмерный массив — массив массивов массивов

```
T = <<R0, C0, D08>, <R0, C1>, . . . , <R2, C2, D08>>
```

где R, C и D — тоже массивы. Для этого T заводится соответствующая константа вне класса, скажем:

```
Array<Vector<Array<uint, 3>>, 8>
_CHECKING_TABLE
```

здесь Vector — массив переменной длины, а капсом — потому что это константа. В Python и Array, и Vector — список.

Для вычисления победителя заводятся метод и поле.

```
private field uint _winner
private _update_winner: uint last_move → void
```

а метод *get_winner* превращается в простой геттер, возвращающий это поле.

В *_update_winner* мы достаём набор “троек” из таблицы и проверяем каждую. Для большего удобства стоит модифицировать метод *_check_line*, чтобы он принимал $\langle i, j, k \rangle$:

```
private _check_line: Array<uint, 3> line, uint winner → uint r
```


не забываем, что первые 4 хода ничего проверять не нужно, а на 7-м выдавать ничью, если победителя так и не появилось.

```
private _no_winner_yet: void → bool r
  r ← _winner = WINNER_NONE

private _update_winner: uint last_move → void
  bool need_check ← _no_winner_yet()
    ∧ _moves_count > 4

  if (need_check)
  then:
    var lines = _CHECKING_TABLE[last_move]
    foreach line from lines
    do:
      _winner ← _check_line (line, _winner)

  bool draw ← no_winner_yet() ∧ _moves_count = 7
  if (draw)
  then:
    _winner ← WINNER_DRAW
```

Это решение задачи приемлемо, оно приводит к наименьшему количеству вызовов `_check_line` и просто для понимания. Но лично мне было бы лень писать громоздкую `_CHECKING_TABLE`.

i, j	C_0	C_1	C_2
R_0	0, 0	0, 1	0, 2
R_1	1, 0	1, 1	1, 2
R_2	2, 0	2, 1	2, 2

Если поглядеть на игровое поле в квадратных координатах, то координаты клетки $\langle i, j \rangle$ однозначно указывают, какие строку и столбец надо проверять — это R_i и C_j . Всего-то и нужно что перевести номер клетки n в квадратные координаты!

	0	1	2	j
0	0	1	2	→
1	3	4	5	
2	6	7	8	
i				↓

$$i = [n / 3]$$

$$j = n \% 3$$

где $[]$ — это целая часть от деления (читается “антье”: $[x]$ — антье икс). Далее, любое R_i и C_j это

$$R_i = \{ i \cdot 3; i \cdot 3 + 1, i \cdot 3 + 2 \}$$

$$C_j = \{ j, j + 3, j + 6 \}$$

что несложно доказать.

Осталось разобраться с диагоналями.

Вариант 1: проверять их всегда. Ну а что? Их всего две.

Вариант 2: захардкодить соответствие между ходом и диагональю.

Вариант 3: родить правила вычисления какую всё-таки диагональ чекать.

Идеальный аналитический вариант.

$$\begin{aligned} n = 4 &\Rightarrow \text{check}(D08, D62) \\ n = 0 \vee n = 8 &\Rightarrow \text{check}(D08) \\ n = 2 \vee n = 6 &\Rightarrow \text{check}(D62) \end{aligned}$$

Понадобится *if / elif / elif* и четыре строки на сами проверки.

Вариант 4: посмотреть, когда точно НЕ нужно проверять диагонали.

Когда?

Ответ: когда центр поля не занят.

Опишем на псевдокоде всё решение с четвертым вариантом проверки диагоналей. Модифицируем сначала `_check_line`, пускай он сразу работает с приватным полем `_winner`.

```
private _check_line: uint i, j, k → void
```

Тогда

```
private _update_winner: uint last_move → void
  bool need_check ← _no_winner_yet() ∧ moves_count > 4
  if (need_check)
  then:
    uint r = [last_move / 3] · 3
    uint c = last_move % 3
    _check_line(r, r+1, r+2)
    _check_line(c, c+3, c+6)
    uint center = 4
    if (get()[center] ≠ CELL_NONE)
    then:
      _check_line(0, 4, 8)
      _check_line(2, 4, 6)
    bool draw ← _no_winner_yet() ∧ _moves_count = 7
    if (draw)
    then:
      _winner ← WINNER_DRAW
```

всего на 5 строк длиннее первого решения (если не учитывать заполнение массива массивов массивов!). На практике четвёртый вариант диагоналей аналогичен первому, так как все всегда стремятся сразу занять центр.

Если ваше решение похоже на одно из этих — похвалите себя! Если нет — не расстраивайтесь: реализуйте на ваш вкус что-то из предложенного.

И вот теперь-то точно УРА! **хо** 0.1 готов!

Занятие 75. Тестовое. Карточки #1

Далее представлены темы, по которым вы должны рассказать всё, что знаете; сами себе. Если ничего не знаете — ничего страшного: узнаете ответ и повторите попытку завтра.

Для усвоения вам могут помочь карточки. Карточка — листочек бумаги, лучше плотной. С одной стороны вы пишете/печатаете тему, с другой — конспектик-ответик. Взяв стопку карточек в руки вы по теме вспоминаете ответ (можно письменно, можно устно). Если не помните чего-то — переворачиваете карточку. Если уверены, что всё вспомнили верно — не переворачивайте карточку, просто положите в конец стопки.

Проверять себя по карточкам можно в любое время дня и месте! Возьмите карточки в метро, на прогулку, на работу — смена обстановки положительно повлияет на усвоение.

Карточки — отличный способ запоминания материала, возьмите его себе на вооружение.

Часть А. Что такое:

- Отношение.
- Связь между Декартовым Произведением и отношением.
- Функциональное отношение.
- Обратная функция.
- Упорядоченное множество.
- Рациональное число.
- Вещественное (действительное) число.

Часть В:

- Представьте как множество $\langle a, b, c \rangle$.
- $f = \{ \langle a, b \rangle, \langle b, b \rangle \}$, $g = \{ \langle a, b \rangle, \langle b, c \rangle, \langle c, a \rangle \}$
 - $D(f)$ — ?
 - $R(g)$ — ?
 - $g(f(a))$ — ?
 - g^{-1} — ?
- $S(\equiv)$ — ?
- $A = \text{Мышь цветная, кошь большая, значит зонтик из Китая. } S(A) \text{ — ?}$
- $S(a \vee (b \wedge \neg c \rightarrow \neg a)) \text{ — ?}$

- D="-", S="ПАУК", L="", C"КОТ".
A=C..L..D..S..L..
- Что за слово A?
- В каком алфавите слово A?

Занятие 76. Тестовое. Карточки #2

Часть А:

- Функция в языках программирования.
- Перечисление.
- Класс. Поле. Метод. Конструктор.
- Наследование

Часть В:

- Базовый класс.
- Абстрактный класс.
- Циклы.
- Разница между присвоением значения, равенством и конструированием объекта.

Часть С:

- Двоичный код.
- Приватные методы и поля. Инкапсуляция.
- Интерфейс.
- Архитектура.

Занятие 77. Стек и куча. Задание

Узнайте о памяти программ. Что такое стек? Какого он обычно размера?
Что такое куча? Чем отличается переменная на стеке от переменной на куче?
Как происходит создание и удаление переменной на куче?