Programming Fundamentals w/ Python

Programming Fundamentals w/ Python

This is your intro to Python.

Python is good for so many applications:

- Robotics
- Cybersecurity
- Apps w/ GUI
- General Programming
    - Learning
    - Competitive

Future Worksheet Ideas

DO MORE ON PAPER!

- Guided Notes
- Worksheets
- Flowcharts & Storyboarding
- Word Problem Abstraction
    - Provide a word problem
    - Have students mark up the word problem with important points
        - Yellow = data ("inputs")
        - Blue = outputs
        - Number tasks
    - Document what wasn't said (Equations that solve the problem)
    - List steps to solve the problem
    - Break "complex" steps into multiple simpler steps

Data Types

Variables & Naming Conventions

- Naming Syntax
- Naming Conventions
- Vocabulary (declare, Initialize)
-

Conditionals

Conditionals - Nesting

Logical Operators

- When multiple conditions result in the same "then" statement, they can be combined using "or" operators.

  If (condition1)
  Then do x
  Else if (condition2)
  Then do x
  Else if (condition3)
  Then do x

  Can be combined to:

  If(condition1 OR condition2 OR condition 3)
  Then do x

  There are some tricks to this when there are multiple "then" options

  Analyze the Leap Year problem: see it as distinct if statements; see it as combined and/or statements
- When nesting conditions, they can be combined using "and" operators:

  if(condition1)
  Then if (condition2)
  Then do x

  Can be combined to:

  if(condition1 and condition2)
  Then do x

For Loops

For Loop Nesting

For loop Strings

While Loop

While Loop Nesting

While Loop Strings

Mixed Loops (For Loops & While Loops)

- When to use each loop

Unit 1 - Core Ideas

Core Ideas

This unit will help students learn the core concepts needed to begin learning python:

- Printing
- User Inputs
- Variables
- Data Conversion
- Operators

01_UserInput.py

The following program walks students through the process of printing information to a screen and begin getting user responses.

- Variable Basics
- Print Basics
- F-Strings
- Escape Sequences

```
#01_UserInput.py
name = input("What is your name?")
print(f"Welcome {name}!")
'''
    To get user input:
        varName = input("_________") → Blank is your
question/request
'''
#Get the user's favorite ice cream flavor, and respond
with:
# "Woah Jack, I love ________ also!"
flavor = input("What is your favorite ice cream flavor?")
print(f"Cool {name}, I love {flavor} also!")
'''
    Print the following message using 1 print statement:
    I bet you can't say "Wee Wee Wee" thrice!
    8\
'''
print("I bet you can't say \"Wee Wee Wee\" thrice!\n8\\")
'''
    Escape Sequences!
    \\ → \
    \" → "
    \' → '
    \n → new line
    \t → tab (indentation)
'''
```

✏️ 02_Drawing.py

Drawing Assignment

This assignment challenges students to use print statements to print ASCII art to the console.

- Print statements
- Escape Sequences

```
#02_Drawing.py


# You are to draw the following using print statements
#     ,     ,
#     (\____/)
#      (_oo_)
#        (O)
#      __||__    \)
# []/______\[] /
# / \______/ \/
```

03_Story.py

This program emphasizes pulling multiple values from the user and plugging them into a single print statement.

- Print statements
- Input statements
- F-Strings

```
restaurant = input("Favorite Restaurant: ")
drink = input("Favorite Drink: ")
food = input("Favorite Food: ")
print(f"Bobby went to {restaurant} to get a {drink} and
{food}.")
```

✏️ 04_MadLib.py

MAD Libs Assignment

This assignment challenges students to pull multiple values from the user and plug them into 1 or more print statements, developing a MadLib story.

- Print statements
- Input statements
- F-Strings

```
'''
    "Give me a name"
    "Give me a word that describes how big something is"
    "Give me a type of animal"....

    [name] had a [size] [animal], its fleece was [color]
    as snow



    print the story with their inputs.

    * Get 5+ inputs from the user
    * plug them into a MadLib story
'''
```

05_Numbers.py

This program

```
#05_Numbers.py
num = 13
num = num + 5   #num → 18
print(num)
num = num - 3   #num → 15
print(num)
num = num * 2   #num → 30
print(num)
num = num / 3   #num → 10
print(num)
num = num // 3   #num → 3   integer division → floor →
round down to the lower whole #.
print(num)
print(.1+.1+.1)
print(4**3)     # prints → 64 → "To the power of"
print("Hi "*7)   #txt * # → repeat text that many time

num += 5        #increase by    → 8
num -= 1        #decrease by    → 7
num *= 2        #multiply by    → 14
num /= 3        #divided by     → 4.6666
print(num)

age = int(input("How old are you?\n"))
age += 5
print(f"In 5 years you'll be {age} years old.")
#int("text") converts text to a whole #
#float("text") converts text to a real #


#Python processes whole numbers differently than real
numbers
#int → integer → whole numbers → 1, 2, 3, 4, 5
#float → floating point → real numbers → 2.55, 37.999,
100.0
```

[redacted]

06_Pythag.py

```python
#06_Pythag.py
'''
    Pythagorean Theorem: a**2 + b**2 = c**2
    Write a program that takes in 2 sides of a right
triangle from the user
    and calculates the hypotenuse (side c)
'''
import math
a = float(input("What is side A's value?"))
b = float(input("What is side B's value?"))
c = math.sqrt(a**2 + b**2)
print(f"{a}**2 + {b}**2 = {c}**2")
```

07_Rectangle.py

```
'''
    Write a program that:
    1. Gets a width (of a rectangle) from the user (and
convert to float/int)
    2. Gets a height (of a rectangle) from the user (and
convert to float/int)
    3. Calculates the Perimeter  2a + 2B
    4. Calculates the Area         a * b
    5. Produces a USEFUL output

    3 Phases of a program:
    * Get inputs
    * Calculate info
    * Show Answer
'''
width = float(input("What is the width of the rectangle?
"))
height = float(input("What is the height of the rectangle?
"))
perimeter = 2*width + 2*height
area = width * height
print(f"A rectangle with sides {width} and {height} has a
perimeter of {perimeter} and an area of {area}")
```

✏️ 08_MyMath.py

```
'''
    08_MyMath.py
    Pick any useful math algorithm to help the user solve a
homework problem.
    You need to get user inputs, calculate a solution, and
provide a useful response.
    Write a single print statement at the beginning
explaining what your program does to
    the user prior to getting any inputs.
    Example algorithms:
    * BMI Calculator
    * Wind Chill Factor
    * Perimeter + Area of a polygon
    * Halflife of radioactive substance
    * Ohms Law
    * Quadratic Formula
    * Etc.
'''
```

09_Remainder.py

```python
#09_Remainder.py
'''
    10/3 → 3.333333333
    10/3 → 3 R1
    15/2 → 7.5
    15/2 → 7 R1

    20/5 → 4 R0
    20/13 → 1 R7
    31/4 → 7 R3
'''
print(10/3)
print(10//3)    #// is Integer Division
print(10%3)     #% modulus, Modulo, Mod = Remainder after
the division

#1: Odds or Evens → %2
print(14 % 2)       # 0 → Even
print(15 % 2)       # 1 → Odd

#2: Last Digit → %10
print(1425 % 10)        # 5
print(123456789 % 10)   # 9

#3: Last n Digits % 10**n
print(1234 % 100)   #34
print(123456789 % 1000) #789

'''
    %2 → 2 values: 0, 1
    %3 → 3 values: 0, 1, 2
    %4 → 4 values: 0, 1, 2, 3
    %n → n values: 0, 1, 2... n-1
'''
```

10_FormattingNumbers.py

```python
#10_FormattingNumbers.py
num = 13.6
print(f"The special number is {num}")
num = 154.9175634812
print(f"The special number is now {num}")
print(f"Pretty Version: {num:.0f}")
print(f"Pretty Version: {num:.1f}")
print(f"Pretty Version: {num:.2f}")
print(f"Pretty Version: {num:.3f}")
print(f"Pretty Version: {num:.4f}")

print(.1+.1+.1)
num = 0
for i in range(20):
    num += .1
    print(f"{num:.1f}")
```

Unit 2 - Conditionals

Conditionals

Conditionals are a major component of any program. It allows a user to tell the code to branch in 1 direction or the other.

Major topics include:

- Booleans - True/False values
- Comparison Operators:
    - <
    - <=
    - >
    - >=
    - ==
    - !=
- Logical operators
    - !
    - and
    - or
- Boolean Logic -

11_Conditionals.py

```python
#11_Conditionals.py
'''
    Conditionals → If Statements
        * Any comparison between 2 or more values
        * Usually involves if, elif, else statements
    Boolean → True/False
    Boolean Logic → Comparisons that evaluate to True/False
    Relational Operators:
        - >     Greater Than
        - >=    Greater Than or Equal To
        - <     Less Than
        - <=    Less Than or Equal To
        - ==    Is Equal To (Check the Value)
        - !=    Is Not Equal To
    Logical Operators
        - and   returns true if 2 conditions are BOTH true
        - or    returns true if ANY 1+ conditions is true
        - not   opposite of current boolean value
'''
#Boolean Values:
isWarm = False
isSunny = False
#Comparisons
age = 15
canDrive = age >= 16
print(f"Old enough to drive: {canDrive}")

age = 8
height = 15
canRideRollerCoaster = age >= 8 and height >= 36
print(f"Can Ride: {canRideRollerCoaster}")

if canRideRollerCoaster:
    print("Yay! Come take a seat!")
else:
    print("Sad day! Maybe next year.")
```

12_If.py

```python
#12_If.py
name = input("What is your name?")
#If the user's name is Tim, print "You rock [name]!"
if name == "Tim":
    print(f"You rock {name}!")
#if the user's name is "Frederick", print "You smell funny [name]."
if name == "Frederick":
    print(f"You smell funny, {name}.")
#if the user's name is _____ or _____, print "I like you lots, [name].
if name == "Kim" or name == "Teddy":
    print(f"I like you lots, {name}... 8)")
    #Notice, you have to re-specify after the "or" statement what you'
    #Python does not assume that you're still comparing the name.
print(f"Have a nice day, {name}.")
```

13_Random.py

```python
#13_Random.py
'''
    1. Get the user's guess
    2. "flip" a coin → random value tails = 2, heads = 1
    3. either need to conver the user's guess to a number, or the computers numbe
    4. if answer is the same as the flip: print "Correct"
    5. otherwise: print "Wrong!"
'''
from random import randint
print("We're going to play a game - \"Heads\" or \"Tails\".")
guess = input("Heads or Tails?")    #Assume that the user only gives valid inputs
flip = randint(1, 2)
if flip == 1:
    flip = "Heads"
else:
    flip = "Tails"
print(f"You guessed {guess}, I flipped {flip}")
guess = guess.lower()
flip = flip.lower()
if guess == flip:
    print("Aw man, you got it right.")
else:
    print("Sucker! You guessed wrong!")

'''
    To use random #'s
    1. import the random library (or a specific function)
        a. import random → random.randint(low, high)
        b. from random import randint → randint(low, high)
    2. create a variable and assign it a random value
        dice = randint(1, 6)
'''
```

14_Winner.py

```
#14_Winner.py
'''
    Roll 2 dice, add their values, and do 1 of the following.
    a. print "You win double!" if you roll equal values
    b. print "You win!" if your roll's sum is 3, 4, 10, 11
    c. print "you lose" if roll is 5, 6, 7, 8, 9
    (will print the first correct option and skip the rest)

    1. Useful print statement (explain the game)
    2. roll 2 separate 6-sided dice (2 variables)
    3. calculate their sum (store in another variable)
    4. process the if/elif/else statements

    if sum == 3 or sum == 4 or sum == 10 or sum == 11:
'''
#Students did this initially on their own before being shown a solution

print("we're playing a dice rolling game.")
die1 = randint(1, 6)
die2 = randint(1, 6)
print(f"D1 {die1} - D2 {die2}")
sum = die1 + die2
if die1 == die2:
    print("You win double!")
elif sum <= 4 or sum >= 10:
    print("You win.")
else:
    print("You lose.")
```

✏️ 15_RPS.py

```
PS.py
Paper Scissors

 Ask the user: Rock, Paper, Scissors (assume the user gives valid input)
 generate a random # (1-3) and store it in a variable
 Convert 1, 2, 3 to Rock, Paper, Scissors (Respectively)
 check who wins
  Ties: both have the same value
  Player Win: → P(Rock) and C(Scissors) or P(Scissors) and C(Paper) or P(Paper) and C(Rock)
  Comp Win: → Everything Else (else statement)

andom import randint
```

16_Grades.py

The following code was used to demonstrate (if, if, if) vs (if, elif, else) statements.

- (if, if, if) are all disconnected statements. They could all execute as true.
- (if, else) These are connected statements. 1 of the 2 must execute as true.
- (if, elif, elif) These are connected, but there's no default option (else). 0 or 1 will be true.
- (if, elif, else) These are all connected statements. 1 and only 1 must execute as true.

When an if statement is followed by elif/else statements it "short circuits" after it finds its first true statement, meaning it will skip all remaining connected elif/else statements.

```
16_Grades.py
''
   Your task:
   - get a user's grade (as a percentage) → 75
   - calculate A, B, C, D, F based on their grade
   - print their Letter-grade
''
rade = float(input("What is your grade? (numeric - ie 85)"))
rint(f"You said {grade}")
f  grade >= 90:
   print("A")
lif grade >= 80 and grade < 90:
   print("B")
lif grade >= 70 and grade < 80:
   print("C")
lif grade >= 60 and grade < 70:
   print("D")
lse:
   print("F")
```

```
''
   3 Types of Errors
      1. Syntax error - Broke rules of language (red underline)
      2. Runtime - Breaks during the run of your program (conversion error, x/0)
      3. Logical (Semantics) - Unexpected Result.
''
```

17_TempFeel.py

Initial Instructions

```
eel.py

 Get a temperature from the user, then determine whether that temp is hot, cold, or warm.
ndition: All temperatures will be in Fahrenheit.
s: hot - anything above 75
   cold - anything less than 65
   warm - anything else
le Dialog:
rint - What is the current temperature in F?
ser Responds - 49
rint - "Wow, that's chilly!"
Level
 Add additional statements for really hot (above 95) and really cold (below 45)
 Could add Freezing, danger of death, and Boiling as well.
```

Code walked through in class:

```
el.py

Get a temperature from the user, then determine whether that temp is hot, cold, or warm.
dition: All temperatures will be in Fahrenheit.
: hot - anything above 75
  cold - anything less than 65
  warm - anything else
```

```
e Dialog:
int - What is the current temperature in F?
er Responds - 49
int - "Wow, that's chilly!"
evel
Add additional statements for really hot (above 95) and really cold (below 45)
Could add Freezing, danger of death, and Boiling as well.


at(input("What is the temperature? (Fahrenheit)"))
95:
f"{temp} is too hot.")
> 75:
f"{temp} is Hot")
>= 65:
f"{temp} is Just right")
>= 45:
f"{temp} is Cold")


f"{temp} is Too Cold!")
```

18_VotingAge.py

Initial Code

```
#18_VotingAge.py
#1. Get an age from the user


#2. Ask the user if they're a citizen (y or n or Yes or no)




#3. Determine whether they can vote
#   → You can vote if you are a citizen and you're at least 18 years old



#4. Otherwise tell them they don't meet the qualifying criteria
#   → All invalid inputs should be considered as though they typed "no"
```

Typical 1st attempt by students:

```
#18_VotingAge.py
#1. Get an age from the user
age = int(input("How old are you?"))
#2. Ask the user if they're a citizen (y or n or Yes or no)
citizen = input("Are you a citizen (Yes, y, No, n)")
citizen = citizen.lower()
#3. Determine whether they can vote
#   → You can vote if you are a citizen and you're at least 18 years old
```

```
if age >= 18 and citizen == "yes" or citizen == "y":
    print("You can vote.")
#4. Otherwise tell them they don't meet the qualifying criteria
#   → All invalid inputs should be considered as though they typed "no"
else:
    print("You don't meet the qualifications to vote.")
```

The code above is 99% correct, but there's 1 bug that leads to a Logical Error.

- Logical errors occur when the program runs, doesn't crash, but your result isn't what you expected.
- "Logical Errors = Unexpected Results"

This code appears to work correctly, but in at least 1 situation you'll get bad results. Depending on how you write the if statement, you may have to type something different, but if yours is typed like mine is, you get the wrong results if the age is less than 18 but the user typed "y"

Why?
The reason has to do with the Order of Operations.

- You know the Order of Operations as PEMDAS
- In computer science, there are more that you need to be aware of:
    - () are highest precedence (still)
    - And → essentially viewed as * (in terms of precedence)
    - Or → essentially viewed as + (in terms of precedence)
    - You can get more details on the full list of operator precedence from GeeksForGeeks

In the code above, the age >= 18 and citizen == "yes" happens first:

Order of this if statement

	Initial values: • age → 13 • citizen → "y"
Initial Code	`if age >= 18 and citizen == "yes" or citizen == "y":`
Substitution	`if 13 >= 18 and "y" == "yes" or "y" == "y":`

Evaluate individual blocks	`if False and False or True:`
Evaluate AND statements	`if False or True:`
Evaluate OR statements	`if True:`

Because the or is the last thing evaluated, if the right side is true it overrides everything on the left side.

To fix this situation we have to use parenthesis. See how the results change:

	Initial values: • age → 13 • citizen → "y"
Initial Code	`if age >= 18 and (citizen == "yes" or citizen == "y"):`
Substitution	`if 13 >= 18 and ("y" == "yes" or "y" == "y"):`
Evaluate individual blocks	`if False and (False or True):`
Evaluate Parenthesis (and the OR inside)	`if False and (True):`
Evaluate AND statements	`if False:`

So all you need is to add the (and) around the 2 citizen portions of the if statement (as shown below)

```
#18_VotingAge.py

#1. Get an age from the user

age = int(input("How old are you?"))

#2. Ask the user if they're a citizen (y or n or Yes or no)

citizen = input("Are you a citizen (Yes, y, No, n)")

citizen = citizen.lower()
```

```
#3. Determine whether they can vote
#   → You can vote if you are a citizen and you're at least 18 years old
if age >= 18 and (citizen == "yes" or citizen == "y"):
    print("You can vote.")
#4. Otherwise tell them they don't meet the qualifying criteria
#   → All invalid inputs should be considered as though they typed "no"
else:
    print("You don't meet the qualifications to vote.")
```

19_TempConvert.py

Intro

In this program I'm doing a few things:

1. Showing students that not every tool is used in every program. (initially we won't need to solve this problem using if statements)
2. Showing students that we can package multiple commands into a single function that we can call any time we want.
    a. **Functions are new commands in Python** (or whatever language you are learning).
    b. Functions allow us to **simplify multiple commands into a single new command**
    c. Allows for **code reuse**
    d. Makes code more **readable**
    e. Makes code **shorter/cleaner**

Starter Code

```
#19_TempConvert.Py
'''

    Not every program needs every tool.

    We can package chunks of code together to create functions.


    Write code that converts a temperature in F to C

        (F - 32) * 5/9 → C
'''
```

Potential Student Solution

```
F = float(input("What is the temperature (in Fahrenheit)? I will convert it to Celsius."))

C = (F - 32) * (5/9)
```

```
print(f"{F}f is {C}c")
```

Note 1: Using single character variable names is generally discouraged because it's not usually readable or obvious what the variables represent. There are a few exceptions to this:

1. (x, y) → calling these anything else just doesn't make sense.
2. Math equations that use single-character variables in the equation to begin with
    a. $a^2 + b^2 = c^2$
    b. $y = mx + b$

In this program we're making use of the math equation exception, as it's pretty obvious in this program that F and C are Fahrenheit and Celsius (respectively).

Note 2: Capital letters are usually discouraged as a first letter for variable names (this usually represents objects which you'll learn about in a future class), and entire words capitalized usually represents a constant (a variable whose value won't change while the program is running, such as PI or INTEREST_RATE).

Function Template:

```
def funcName(neededItem1, neededItem2):
    command1
    command2
    ...
    return result (if needed)
```

Important information regarding Functions:

- Also called "Methods", "Subroutines"
- 1st line is called the Method Signature
    - def = define (we are telling python we are defining a new command)
    - funcName = the name of the function (what do you want to call it)
        - When you want a dog to sit you say "Sit", so sit would be a good name for that command.
    - In the parenthesis goes items the function needs to work. We'll keep this empty for now.

There are a few different types of functions:

- Functions that take in 0 values and return 0 values
- Functions that take in 1+ values and return 0 values
- Functions that take in 0 values and return 1 value
- Functions that take in 1+ values and return 1 value
  (notice, functions can take in multiple values but can only return 0 or 1 value).

For this lesson we only care about taking in 0 values and returning 0 values.

If we wanted to package the 3 lines that make up our program into a reusable command, here's how we'd do it:

```
ef FtoC():     #Function definition    (1st line is also called "function signature")
  F = float(input("What is the temperature (in Fahrenheit)? I will convert it to Celsius."))
  C = (F - 32) * (5/9)
  print(f"{F}f is {C}c")
```

This defines the function, but alone it's no better than a recipe… with a recipe you can make food, but the recipe alone doesn't automatically give you the food.

To use this function we need a **call statement**

```
FtoC()  #call statement
```

Here's what our new code looks like:

```
FtoC():     #Function definition    (1st line is also called "function signature")
F = float(input("What is the temperature (in Fahrenheit)? I will convert it to Celsius."))
C = (F - 32) * (5/9)
print(f"{F}f is {C}c")


()  #call statement
```

Does this really improve our program?

- Does it make the code shorter? Doesn't look like it.
- Does it make it cleaner? Well, it's organized. I guess…
- Is it more readable? Hard to tell in this program.
- Can we reuse the code? Yes.

As is, this code doesn't seem like an improvement, but what if you needed to convert 5 temperatures? Check this out:

```
FtoC():     #Function definition    (1st line is also called "function signature")
F = float(input("What is the temperature (in Fahrenheit)? I will convert it to Celsius."))
C = (F - 32) * (5/9)
print(f"{F}f is {C}c")


()  #call statement
()  #call statement
()  #call statement
()  #call statement
()  #call statement
```

I simply added 4 new lines that call the FtoC function. Without separating it into its own function I'd have had to call the same 3 lines over and over again (12 lines of code instead of 4)

So, Does this really improve our program?

- Does it make the code shorter? Yes (if we call the function multiple times)
- Does it make it cleaner? This is easier to tell when we have multiple functions.
- Is it more readable? Also hard to tell in this small of a program, but yes it does.
- Can we reuse the code? Yes.

Now, we can take it to the next level and do the following:

- Modify the code to start by asking the user if they want to start with Fahrenheit or Celsius as their temperature.
- Upon getting response check to see if their response was:
    - Fahrenheit (or starts with F)
    - Celsius (or starts with C)
    - Invalid (anything else)
- Call the appropriate function if the user typed Fahrenheit or Celsius (that means we now need 2 different functions), otherwise print "I didn't understand that response"

20_Mathy

#20_Mathy.py

Just like in the previous program, we want to create a chatbot that does a couple simple things.
Ask the user if they want to add, pythag, or roll a die
Using if statements, figure out which option they said and respond appropriately:
Example:

"Welcome to my program"
"Would you like to add, pythag, or roll?"
add
"Number 1: "
5
"Number 2: "
7
"Result: 5 + 7 = 12

Adding should ask for 2 numbers and print their sum.
Pythag should ask for 2 numbers (sideA and sideB) and calculate the hypotenuse

$sideA^2 + sideB^2 = sideC^2$

Roll should get a random # from the user (1-6) and print the result

Alternatively you can roll 2 dice, print their values, and print their total.

Unit 3 - For Loops

21_Redundant.py

This is to prepare us for loops

```python
#21_Redundant.py
numValues = int(input("How many #'s do you have? I'll calculate the Average."))
#get num1 from the user
num1 = int(input("Num: "))
#get num2 from the user
num2 = int(input("Num: "))
#get num3 from the user
num3 = int(input("Num: "))
num4 = int(input("Num: "))
num5 = int(input("Num: "))
num6 = int(input("Num: "))
num7 = int(input("Num: "))
num8 = int(input("Num: "))
num9 = int(input("Num: "))
num10 = int(input("Num: "))

#ave → sum of (num1, num2, & num3) divided by 3
ave = (num1 + num2 + num3 + num4 + num5 + num6 + num7 + num8 + num9 + num10) /
10
print(ave)
```

What if you needed to get 100 #'s from the user?

- How many variables would you need?
- How tedious would the "ave = (num1 + num2…)" line be?

22_ForLoops.py

Python

```
#22_ForLoops.py
"""
    For loops:
        1. Numbered loops or collection loops
            - loop 10 times
            - loop through all the items in a list
        2. Don't use when you don't know how many times you're looping.
        3. Template for a for loop
            for i in range(#):
                do something
                do something
            no longer looping
"""
numValues = int(input("How many numbers do you have? I'll calculate the
average."))
total = 0
for i in range(numValues):
    #get 1 number from the user
    num = int(input(f"Num{i+1}: "))
    #add it to a total
    total += num
print(f"Sum: {total}")
#calculate the average
ave = total / numValues
#print the average
print(f"Ave: {ave}")
print(f"Ave: {ave:.2f}")
```

23_SimpleForLoop.py

Python

```python
#23_SimpleForLoop.py
#Your task: create a for loop that loops 10 times, printing your counter (i).
(prints 0-9)
for i in range(10):
    print(i)
print("***************")
#Task: Create a for loop that prints 7 times, printing Count Von Count's
counting to 12
#   ("1, Ah ah ah", "2, Ah ah ah", ..."12, Ah ah ah")
# - Sesame Street Reference
for i in range(12):
    print(f"{i+1}, Ah ah ah")
    #exact same loop
for i in range(1, 13):
    print(f"{i}, Ah ah ah")
print("***************")
#Task: Count by 2's, starting at 0. {0, 2, 4, 6, ...} all the way to 16
(inclusive).
for i in range(9):
    print(f"{i*2}")
#exact same loop:
for i in range(0, 17, 2):
    print(i)
#start i at 0, while i is less than 17, increase i by 2
#loop from 0 to (less than) 17, increasing by 2
print("***************")
#print the values 5, 10, 15, 20... 50 using a for loop.
for i in range(10):
    print(f"{(i*5)+5}")
for i in range(1, 11):
    print(f"{(i*5)}")
for i in range(5, 51, 5):
    print(i)
```

24_runners.py

Initial code/instructions

```python
#24_runners.py
#   (as in running)
"""
    Your task:
    * Ask the user how many racers were running.
    * Loop that many times
        * Get a runner's time from the user (in seconds)
        * calculate the fastest time (so far)
        * calculate the slowest time (so far)
    * print the results.
    Variables:
    * numRunners → user's input
    * currentRunner → user's input (in a loop)
    * slowestTime → 0 (Precondition: assume no runners can run faster than
this)
    * fastestTime → 1000 (Precondition: assume no runners take longer than
this)
    Functions
    * min(#, #) → gets the smallest value
    * max(#, #) → gets the largest value
"""
```

Solution:

```python
numRunners = int(input("How many runners?"))
fastestTime = 1000
slowestTime = 0
for i in range(numRunners):
    currentTime = int(input(f"Runner #{i+1}'s Time: "))
    fastestTime = min(currentTime, fastestTime)
    slowestTime = max(currentTime, slowestTime)
    print(f"Fastest Time So Far: {fastestTime}")
    print(f"Slowest Time So Far: {slowestTime}")
print(f"Fastest Time: {fastestTime}")
print(f"Slowest Time: {slowestTime}")
```

- The 2 print statements in the for loop are unnecessary, but they show what happens every iteration of the program.

25_Temps.py

Initial Code/Instructions

```python
#25_Temps.py
"""
    Get a starting temperature (in celsius) from the user.
    Get an ending temperature (in celsius) from the user.
    Get a "step" value from the user
    Print every celsius temperature (inclusive) in the provided range.
    *** If the ending value isn't landed on by the step, don't print it
    Also, print the fahrenheit equivalent of each celsius temperature.
"""
```

Solution:

```python
startTemp = int(input("Starting Temp: "))
endTemp = int(input("Ending Temp: "))
step = int(input("Step: "))
for celsius in range(startTemp, endTemp+1, step):
    fahrenheit = celsius * (9/5) + 32
    print(f"{celsius}c = {fahrenheit:.1f}f")
```

26_FizzBuzz.py

Python

```
#26_FizzBuzz.py
"""
    What we know:
    - Save data by creating variables
    - Convert data (str→int)
    - Inputs/Outputs + formatting strings
    - Random #'s
    - Math Stuff
    - Conditionals
    - For Loops (#'d Loops)

    FizzBuzz:
    - Get 2 #'s from the user (fizz, buzz)
    - the computer starts counting from 1 to 100, except
        - if the # is divisible by fizz, prints "fizz"
        - if it's divisible by buzz, prints "buzz"
        - if its divisible by both, "fizzbuzz"
        - otherwise it prints the #
    % = Modulus = Remainder
        7%3 → 1 (remainder of 1 → 2R1)
        8%3 → 2
        9%3 → 0
"""
fizz = int(input("Fizz: "))
buzz = int(input("Buzz: "))
for i in range(1, 101):
    if i % fizz == 0 and i % buzz == 0:
        print("fizzbuzz")
    elif i % fizz == 0:   #i is divisible by fizz
        print("fizz")
    elif i % buzz == 0:
        print("buzz")
    else:
        print(i)
```

27_LuckyGuess.py

Python

```
#27_LuckyGuess.py
"""
    play heads or tails 10x.
        - get a guess from the user ("Heads", "Tails")
        - Get a random # from the computer (0, 1)
        - Convert the computer's number to "Heads"/"Tails"
        - Display the 2 values ("Player guessed ____ and the answer is ____")
        - Tell the user whether they're correct or wrong
    Calculate how many times the user guessed correctly
        Increment your counter IF the user was correct
    Calculate the percentage of correct guesses.
        percent correct is # of correct guesses divided by the # of total
guesses * 100
    *** Try this without looking at your heads/tails program
"""
from random import *
count = 0
for i in range(10):
    print(f"Game #{i+1}")
    guess = input("Heads/Tails: ")
    flip = randint(0, 1)
    if(flip == 1):
        flip = "Heads"
    else:
        flip = "Tails"
    print(f"Player guessed {guess} and the answer is {flip}")
    if guess == flip:
        print("Correct!")
        count += 1
    else:
        print("Wrong!")
print(f"You guessed correctly {count} times")
percent = count / 10 * 100
print(f"{percent:.0f}%")
```

28_Goat.py

This program was scrapped - I realized as we started it that we didn't have all the info needed to complete this coding challenge.

29_Square.py

Python

```
#29_square.py
"""
    Write code that prints the following:
    ***
    ***
    ***
    instead print an nxn square:
        Ask the user how big the square is and print that # of *'s
        on each row, and that many rows.
"""
size = int(input("How big is your square? I'll print it."))
stars = ""
for i in range(size):
    stars += "* "
for i in range(size):
    print(stars)
```

✏️ 30_StepCounter.py

Sasquatch Step Counter

Story: Deep in the heart of the forest, Sasquatch stumbled upon a curious device—a fitness tracker flung off a hiker's wrist in a moment of panic. Intrigued by this new gadget, Sasquatch decided to use it to track his daily wanderings. However, the tracker had a quirk: it reset at midnight each night due to the rough handling it endured. Determined to know his weekly step count, Sasquatch diligently recorded his steps at the end of each day for a week. Your mission is to help Sasquatch calculate the total number of steps he took over these 7 days.

Objective: Write a program that:

1. Prompts the user to enter the number of steps Sasquatch took each day for 7 days.
2. Calculates the total number of steps taken over the week.
3. Prints the total number of steps.

Example Output:

```
Enter the steps for day 1: 10000

Enter the steps for day 2: 12000

Enter the steps for day 3: 8000

Enter the steps for day 4: 15000

Enter the steps for day 5: 11000

Enter the steps for day 6: 9000

Enter the steps for day 7: 13000

Total steps taken in the week: 78000
```

Hints:

- Use a `for` loop to iterate through the 7 days.
- Use a variable to keep a running total of the steps.
- Make sure to convert the input from the user to an integer before adding it to the total.

✏️ 31_NumberHunt.py

Sasquate Number Hunt

Story: One misty morning, Sasquatch decided to explore a hidden valley deep in the forest. Along the way, he found five mysterious stones, each with a unique number carved into it. Curious about the significance of these numbers, Sasquatch wants to analyze them. He needs to find out the smallest and largest numbers, as well as the sum and average of all five numbers. Your task is to help Sasquatch with this numerical quest.

Objective: Write a program that:

1. Prompts the user to enter five numbers.
2. Finds and prints the smallest number.
3. Finds and prints the largest number.
4. Calculates and prints the sum of the numbers.
5. Calculates and prints the average of the numbers.

Example Output:

```
Enter number 1: 7
Enter number 2: 13
Enter number 3: 5
Enter number 4: 20
Enter number 5: 9
Smallest number: 5
Largest number: 20
Sum of numbers: 54
Average of numbers: 10.8
```

This challenge can be solved without storing the contents in lists and without needing variables for every number received.

You should:

- **Loop 5 times**
  - **Get a number**
  - **Add it to the sum**
  - **Figure out if it's the largest # found so far**
  - **Figure out if it's the smallest # found so far**
- **Calculate the average**
- **Print all the important info.**

✏️ 32_LongPowers.py

Long Powers

Python has a neat shortcut for calculating powers:

- 3**5 → 3 to the power of 5

Even though Python offers this method of calculating powers, not all programming languages do.

You're tasked with creating a program that simulates this method of calculating powers by using a for loop.

Get 2 #'s from the user (base, power)

set an initial product value as 1 (similar to how we set sums to 0)

Loop power times:

Multiply the product by the base.

Example:

Base Input: 7

Power Input: 4

(1st loop: Product → 1 * 7 = 7)

(2nd loop: Product → 7 * 7 = 49)

(3rd loop: Product → 49 * 7 = 343)

(4th loop: Product → 343 * 7 = 2401)

7 to the power of 4 is 2401

Unit 4 - Loopy Strings

33_Strings.py

See W3Schools' [Python Strings]

Not Covered

Not covered, but students should know:

- Quotes in Strings
    - Different Quote Types
        - "....'....'...."
        - '...."...."...'
    - Escape Sequences
        - "....\"....\"...."
        - '....\'....\'....'
- Multiline Strings (we often use these as Python Multi-line Comments)

```Python
"""
	This is a python Multiline String

	It can also be used as a python Multiline Comment

	Same can be done w/ ''', but you're more likely to need apostrophies in comments
	than Quotation Marks.
"""
```

- Strings are arrays - we haven't taught arrays yet, so this may not make much sense yet.
-

Class Code

```Python
#33_Strings.py
"""
    Strings: Lists of Characters
        * ASCII - A set list of ordered characters (ASCII Table)
    Strings have special functions
"""
print(chr(65))
```

```
print(ord("A"))
text = "Hello World!"
print(text.isdecimal())
print(text.isalpha())       #Whether the text contains ONLY letters (no #'s or
symbols)
print(text.find("World"))   #Found @ location 6
print(text.find("world"))   #Found @ location -1 (doesn't exist)
print(text[0])              #Grab character @ location 0 → H
print(text[6])              #Grab character @ location 6 → W
#print(text[40])             #Index out of bounds exception (Runtime Error)
print(text[2:7])           #[start@:endBefore] → llo W
      #(didn't print the o @ location 7)
print(len(text))           #Gets the length of the text
print(text[len(text)-1])   #len(text) - 1 → last index
print(text[4:len(text)])   #Get substring from location 4 to the end
print(text[4:])            #Get substring from location 4 to the end
print(text[0:8:2])         #[start@ : endBefore : skipBy]
      #→ Gets every other character (stopping before 8)
print(text[0: :2])          #print every other character (all the way through)
print(text[1: :2])



#Encoding a String
newText = ""
for symbol in text:
    newText += chr(ord(symbol)+1)
print(newText)
#Can you figure out how to decode the newText so it's back to the original?
```

34_VowelCounter.py

[In the future, leave this off and keep as a graded challenge, as it's similar to the "Character Frequency" problem]

Vowel Counter

- **Challenge:** Write a Python program that takes a string as input and counts the number of vowels (a, e, i, o, u) present in the string.
- **Example:**
  - Input: "Hello World"
  - Output: 3
- **Hint:** Use a `for` loop to iterate through each character of the string, and an `if` statement to check if the character is a vowel.

Python

```python
#Get text from the user
text = input("Give me some text, I'll count how many vowels are there.\n")
#Count How many of the letters are vowels
count = 0
for symbol in text:
    symbol = symbol.lower()
    #if the symbol is a vowel (this could be written as separate if statements)
    if symbol == "a" or symbol == "e" or symbol == "i" or symbol == "o" or
symbol == "u" or symbol == "y":
        count += 1
print(f"There were {count} vowels in {text}")
```

35_PalindromeChecker.py

Palindrome Checker (Limited)

- **Challenge:** Write a Python program that takes a string and determines if the string is the same forwards and backwards.
- **Example:**
    - Input: "madamadam"
    - Output: True
    - Input: "helloolle"
    - Output: True
    - Input: "helloWorld"
    - Output: False
- Assume text will be given without spaces or special characters (all characters will be 1 of the 26 letters of the alphabet - though they may not all be the same capitalization)

Solving with loops

(this is how you'd have to do it in most programming languages)

```Python
text = input("Give me some text. I'll check to see if it's a palindrome.")
text = text.lower()
reversedText = ""
for letter in text:
    reversedText = letter + reversedText
    print(reversedText)
if reversedText == text:
    print("Palindrome!")
else:
    print("Nope! Not a palindrome.")
```

Solving with built-in python String stuff

text[: : -1] gets a reversed version of the text.

```Python
#35_PalindromeCheck.py
#Get some text from the user
text = input("Give me some text. I'll check to see if it's a palindrome.")
text = text.lower()
```

```
if text == text[ : :-1]:
    print("Palindrome")
else:
    print("Not a palindrome.")
```

Bonus Round:

Improve the palindrome checker to account for spaces and special characters:

- Get text from user
- Lowercase the entire text
- Remove all non-alphabetic characters from the String (or store only alphabetic characters into a new String)
- Check if same forwards & backwards.

36_CharacterFrequency.py

Character Frequency

- **Challenge:** Write a Python program that takes a string as input and finds the frequency of a specific character (e.g., 'a').
- **Example:**
    - Input: "banana"
    - Character: 'a'
    - Output: 3

37_AlTeRnAtEcAsE.py

Alternating Case

- **Challenge:** Write a Python program that takes a string as input and returns a new string where the characters alternate between uppercase and lowercase, starting with uppercase.
- **Example:**
    - Input: "hello"
    - Output: "HeLlO"
    - Input: "PYTHON"
    - Output: "PyThOn"

38_WordCount.py

The following challenge, wordSearch (probably more accurate to call word count), searches through some text counting how many words there are and printing the words along the way.

There are loads of better solutions, but they usually include tools we haven't learned yet (String split, for example). For this exercise we are trying to solve the problem using tools we've learned, so stick to looping through every character to find start and ends of words.

```python
#38_WordSearch.py
"""
    Get a sentence from the user.
    Find & count every word in the sentence
    print count# - word

    Example:
        Input: I like cheese
        Output: 1. I
                2. like
                3. cheese
        Input: Pepperoni pizza is the best.
                1. Pepperoni
                2. pizza
                3. is
                4. the
                5. best.
"""
```

Solution:

```python
#Find a space
txt = input("Write a sentence. I'll identify/count all the words.")
#Set variables to represent first letter of a word & space location separating
words
firstLetter = 0
spaceLoc = 0
#Set variables to represent the character location as you loop through a String
& the word count
i = 0
wordCount = 0
#Loop through every letter of a given text
for symbol in txt:
```

```
        #If the symbol is a space
        if symbol == " ":
                #increase the word count
                wordCount += 1
                #identify this location as the space location
                spaceLoc = i
                #Print "#. wordFound"
                print(f"{wordCount}. {txt[firstLetter : spaceLoc]}")
                #prep for next word by changing firstLetter to the location
after this space.
                firstLetter = i + 1
        #increase the letter counter/index
        i+=1
#At the end of our string, we still need to print the final word.
#Increase the word counter
wordCount += 1
#Print the info
print(f"{wordCount}. {txt[firstLetter : ]}")
```

✏️ 39_LambSauce.py

Python

```
"""
    You have been tasked by Ramsey Gordon to find the lamb sauce, but you are
having trouble.
    You need to write a program to determine if the dish you are given has lamb
in it.

    You don't care about punctuation or spaces.
    Valid lamb dishes:
    * Lamb with lemon sauce
    * hello la, mb
    * LaMB sauce
    * l a,m   -   b

    Invalid lamb dishes:
    * llaammbb
    * baml


    Given a line of text (user input)
    Lowercase the text
    Remove all non-letters from the String (or form a new String w/ only
letters)
    Check if the word "lamb" appears anywhere in the text (including within a
different word)

    Helpful functions/tools:
    txt="Hello World"
    (...Lowercase txt)
    for symbol in txt:
        if symbol.isalpha() → tells you whether or not the symbol is alphabetic
(a letter)
            ...if so you can put it in a different piece of text that only
contains letters.

    txt[4] → gives you the character @ location 4
    txt[4:12] → gives you the substring from location 4 through location 11
(stopping before 12)
"""
```

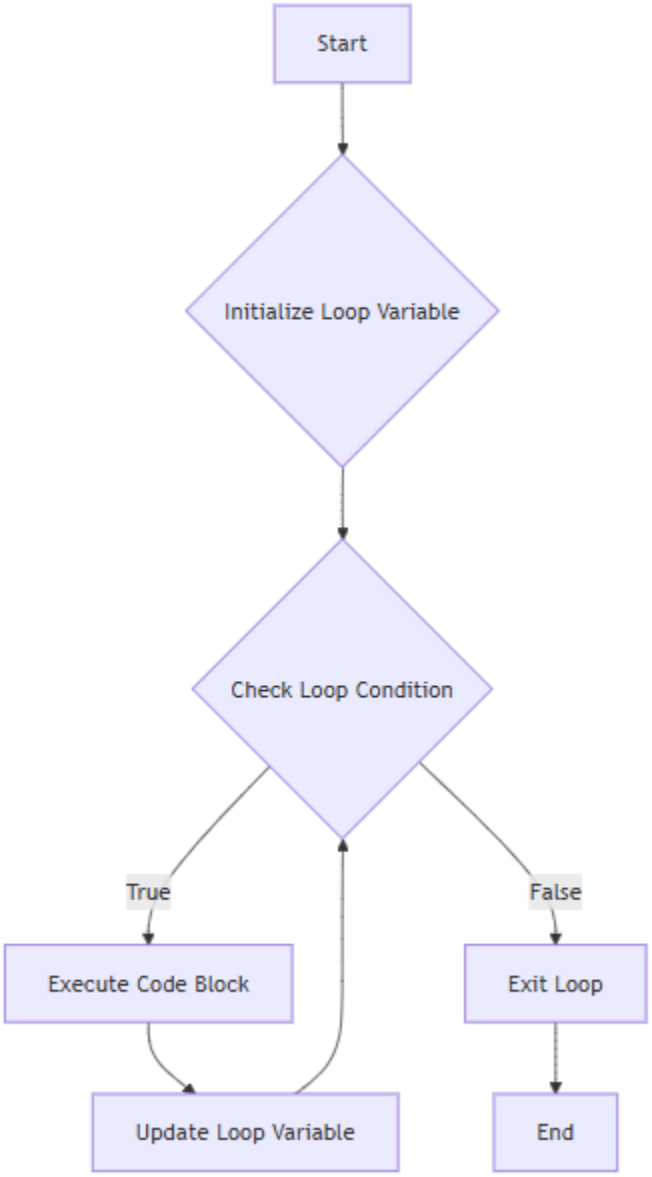
40_WordBalance.py

Python

```
"""
A sentence is said to be balanced if the number of spaces + vowels is the same
as the number of other symbols (including punctuation).
You need to figure out whether or not the characters are balanced.
    - "Hello" → Not Balanced (3 consonants, 2 vowels/spaces).
    - "Here We'll Be" → Not Balanced
    - "AEIOU BCDFGH"> Balanced (5 vowels + 1 space, 6 consonants)
"""
```

Unit 5 - While Loops

While Loops



Cornell Notes: Python `while` Loops

Class: Programming Fundamentals **Topic:** Python `while` Loops **Date:** June 6, 2025

Cue Column (Questions/Main Ideas)	Notes Column (Detailed Information/Examples)
What are `while` loops?	**`while` Loops vs. `for` Loops:** - **`for` Loops:** Used when the number of iterations (loops) is **known** beforehand. - **`while` Loops:** Used when the number of iterations is **unknown**; loops "as long as" a condition remains `True`. - Essentially a **looping `if` statement** – it repeatedly checks a condition.
How do `while` loops work?	**Core Structure:** 1. **Initialize Loop Variable:** A variable is set up *before* the loop starts. This variable controls the loop's condition. 2. **Condition Check:** The loop continues *as long as* the variable meets a specified condition. 3. **Do Something:** Code inside the loop is executed. 4. **Update Loop Variable:** The loop variable *must* be changed *inside* the loop. This ensures the condition eventually becomes `False` and the loop terminates.
What's an "Infinite Loop"?	**Potential Pitfall: Infinite Loops** - Occurs when the loop condition *never* becomes `False`. - This usually happens if the loop variable is not updated correctly, or the condition is always `True`. - The program will get stuck in the loop indefinitely.
Example 1: Countdown	**Code Example 1: Simple Decrement** ```python

loopVariable = 11

while(loopVariable > 0):

loopVariable -= 1

print(loopVariable)

Output: 10, 9, 8, ..., 0

```

- **`loopVariable`** starts at 11.

- Loop continues `while` `loopVariable` is greater than 0.

- `loopVariable` is decremented by 1 each time.

**Example 2: User Input Loop**

**Code Example 2: Interactive Loop**

```python

again = input("Would you like to play?").lower()

while again == "yes":

print("Excellent! (pretend we just did something)")

again = input("Would you like to repeat?").lower()

print("Thanks for playing.")

```

- Loop continues as long as the user inputs "yes".

- The `again` variable is updated inside the loop based on user input.

Example 3: Rolling Dice

Code Example 3: Random Condition

```python

from random import *

d1 = randint(1, 6)

d2 = randint(1, 6)

while d1 != d2:

print(f"You rolled {d1} & {d2}")

d1 = randint(1, 6)

d2 = randint(1, 6)

print(f"You rolled doubles! {d1} & {d2}")

```

- Loops until two random dice rolls are equal (doubles).

- The loop variables (`d1`, `d2`) are updated randomly.

What is a Flag/Sentinel?

`Flag` / `Sentinel` Variable:

- A `boolean` variable (usually `True` or `False`) that controls the loop.

- The loop repeats/stops based on the `True`/`False` state of this variable, rather than a direct condition on changing numeric values.

Example 4: Flag/Sentinel

Code Example 4: Using a Flag

```python

foundDoubles = False

while foundDoubles == False: # As long as foundDoubles is false

d1 = randint(1, 6)

d2 = randint(1, 6)

print(f"You rolled a {d1}, {d2}")

if d1 == d2:

foundDoubles = True # Change flag to True to stop loop

print("Doubles! Hazaa!")

```

- The `foundDoubles` flag is set to `True` only when doubles are rolled, stopping the loop.

Should I use `break` statements?

`break` Statements:

- Immediately **kicks out of a loop** prematurely, regardless of the looping condition.

- **General Recommendation:** Avoid using `break` statements in introductory programming.

- **Why avoid?**

- Misleading: Makes code harder to follow.

- Less readable: Breaks normal flow.

- Less intuitive: Obscures loop logic.

- *Note:* In advanced scenarios, `break` can simplify some problems, but it's best to stick to clear loop conditions for now.

Example 5: `break` (Don't Use!)

Code Example 5: `break` Demonstration (for understanding, not use!)

```python

while(True): # Infinite loop unless broken out of

d1 = randint(1, 20)

print(d1)

if d1 == 7:

break # Exit loop when 7 is rolled

print("Quit because you rolled a 7.")

```

- This loop would run forever (`while True`) if not for the `break` statement when `d1` equals 7.
```

What is Data Validation?

Application: Data Validation with `while` Loops

- A common use for `while` loops.

- **Purpose:** Ensures user input is valid (matches approved options).

- **Mechanism:** The loop repeats *as long as the input is invalid*.

- Once valid input is received, the loop terminates, and the program continues.

Example 6: Data Validation

Code Example 6: Heads or Tails Game (Validation)

```python

# ... (initial setup)

while playGame == True:

# Beginning of Data Validation

option = input(f"Round #{gameCount+1}: Heads or Tails (or 'Done' to quit)?")

while option != "Heads" and option != "Tails" and option != "Done":


print(f"You chose {option} - which is not a valid 'Heads', 'Tails' or 'Done' option.")

option = input(f"Round #{gameCount+1}: Heads or Tails (or 'Done' to quit)?")

# End of Data Validation

print(f"You chose {option}.")

# ... (game logic continues)

```

- The inner `while` loop checks if `option` is "Heads", "Tails", or "Done".

- If invalid, it prints an error and asks again, looping until valid input is given.

AVID Strategies / Thinking Ahead:

- **Self-Questioning:**
  - "When would a `while` loop be better than a `for` loop?" (When you don't know how many times you'll repeat!)
  - "What happens if I forget to update my loop variable?" (Infinite loop! Crash!)
  - "How can I use a `while` loop to make sure someone enters their age correctly?" (Data validation!)
- **WICOR Connection:**
  - **Writing:** The notes themselves, code examples, summary.
  - **Inquiry:** Asking "Why avoid `break`?", "How does `while` differ from `for`?".
  - **Collaboration:** Discussing when to use `while` vs. `for` with peers.
  - **Organization:** Cornell Notes format, graphic organizer.
  - **Reading:** Analyzing the code examples.
- **Common Pitfalls to Remember:**
  - **Forgetting to initialize the loop variable.**
  - **Forgetting to update the loop variable** inside the loop (leading to infinite loops).
  - **Incorrectly setting the loop condition**, causing the loop to run too many or too few times, or never at all.
  - Using `break` statements when a clear condition would suffice.

41_WhileLoops.py

Python

```
"""
    While Loops:
        For Loops - Known # of Loops
        While Loops - Unknown # of Loops

        Loop "as long as" a condition is true

        Looping if statement

        set variable        (The looping variable)
        As long as the variable meets a condition
            do something
            update variable

        Potential: Infinite Loop
"""
from random import *
loopVariable = 11
while(loopVariable > 0):
    loopVariable -= 1
    print(loopVariable)

again = input("Would you like to play?") # yes = repeat, no = stop repeating
again = again.lower()
while again == "yes":
    print("Excellent! (pretend we just did something)")
    again = input("Would you like to repeat?")
    again = again.lower()
print("Thanks for playing.")

d1 = randint(1, 6)
d2 = randint(1, 6)
while d1 != d2:
    print(f"You rolled {d1} & {d2}")
    d1 = randint(1, 6)
    d2 = randint(1, 6)
print(f"You rolled doubles! {d1} & {d2}")

"""
    Flag/Sentinel
        Condition in which you repeat/stop based on a boolean value rather than
        a boolean condition.
"""
```

```
print("********************")
"""
    Goal:
    As long as foundDoubles is false
        roll 2 dice
        print values
        if the values are doubles
            change foundDoubles to true
"""
foundDoubles = False
while foundDoubles == False:
    d1 = randint(1, 6)
    d2 = randint(1, 6)
    print(f"You rolled a {d1}, {d2}")
    if d1 == d2:
        foundDoubles = True
print("Doubles! Hazaa!")
print("**************************")
"""
    Break Statements - kicks out of a loop prematurely (ignoring the looping
condition).
    Don't use break statements because it's
       misleading
       less readable
       less intuitive
     In a future course we'll explore examples where using a while loop or a
for loop
       is less intuitive and problems become easier to solve when using break
statements.
       Until then - don't use them.
"""
while(True):
    d1 = randint(1, 20)
    print(d1)
    if d1 == 7:
        break
print("Quit because you rolled a 7.")
```

42_validate.py

Python

```
#42_validate.py
"""
    Data validation requires repeating as long as the input is invalid (doesn't
match a set of approved options)
    In the following program, the only portion important for this example is
the lines
    between "#Beginning of Data Validation" and "#End of Data Validation".


    Heads/Tails → Continuously repeat asking "Heads or Tails" until the user
types "Done".
        As long as an input is invalid, we'll ask them again before coninueing
with the game.
"""
from random import *
playGame = True
gameCount = 0
while playGame == True:
    #Beginning of Data Validation
    option = input(f"Round #{gameCount+1}: Heads or Tails (or 'Done' to
quit)?")
    while option != "Heads" and option != "Tails" and option != "Done":
        print(f"You chose {option} - which is not a valid 'Heads', 'Tails' or
'Done' option.")
        option = input(f"Round #{gameCount+1}: Heads or Tails (or 'Done' to
quit)?")
    #End of Data Validation
    print(f"You chose {option}.")
    if(option == "Done"):
        playGame = False
    else:
        flip = randint(1, 2)
        gameCount += 1
        if flip == 1:
            flip = "Heads"
        else:
            flip = "Tails"
        if option == flip:
            print(f"{flip} - Correct!")
        else:
            print(f"{flip} - Wrong!")
print(f"You played {gameCount} games with me.")
print("Thanks for playing with me.")
```

✏️ 43_HiLo.py

High Low Game

Guess My Number

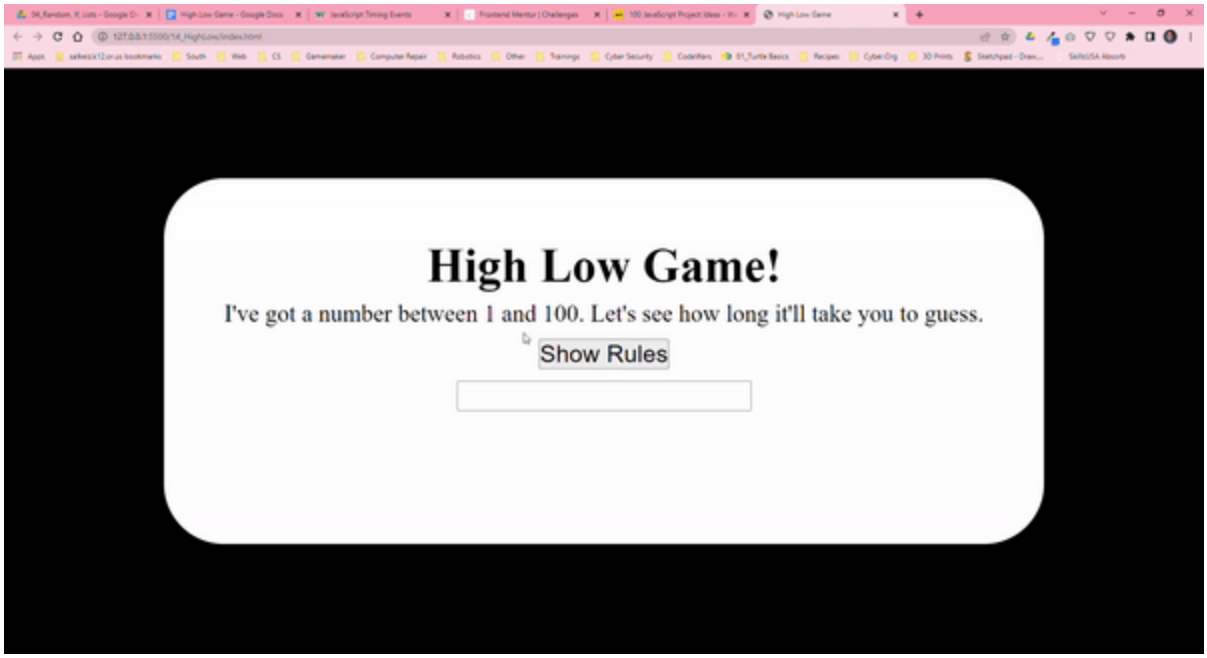
For this assignment, you're going to create a High Low Game.

Basic Rules:

- Computer generates a random whole #
- User guesses.
- Computer responds "Too High", "Too Low" or "Correct"
- When the user is correct, the computer generates a new random number and the game starts over.

Requirements

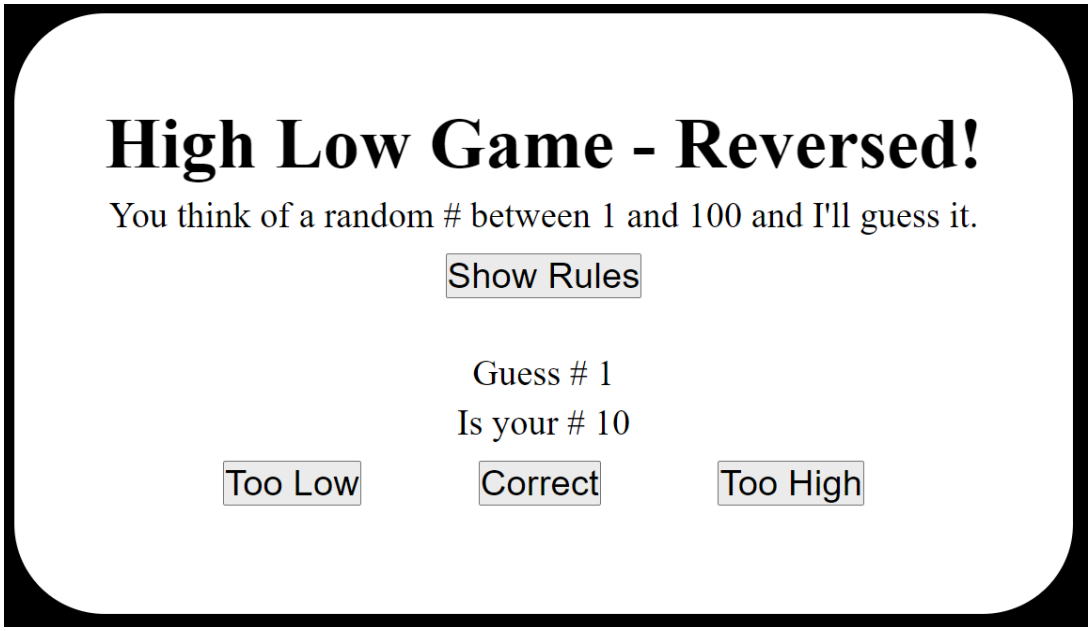
- **Game works as described above (javascript)**
  - When the user makes a guess, clear the input by setting it to "", this way the user doesn't have to erase to guess again.
- HTML & CSS work well and look good.
- Good computer responses during the game.
- Extra Credit:
  - Display the attempt # as you play ("Example - Guess # 3: 57").
  - Epic HTML/CSS or added javascript features (example: modal for rules)
  - High Low Reverse (User thinks of a #, computer tries to guess it)



Scoring

5 points	HTML	Valid HTML
5 points	CSS	Looks nice
10 points	JavaScript	Correctly plays the High Low game (following the basic rules) and provides useful computer responses throughout the game. When the user guesses correctly the computer should generate a new random number and invite the user to play again.
1-? EC	CSS/Javascript	Exceptional Design Track guess number Create the High Low Reverse game (user thinks of #, computer tries guessing).

High Low Reverse:



- In "High Low Reversed", the computer keeps track of a low and a high limits
- As the computer guesses and the user says "Too High" or "Too Low", the computer adjusts its limits.
- The computer can play by efficiency (always guess the midpoint between the high and low limits) or randomly (always guess a random number between the high and low limits).

Example play:
The user is thinking of the number 17.
Computers limits are 1 & 100

- Computer: Is your # 35?
- User: Too High (new limits: 1 & 34)
- Computer: Is your # 8?
- User: Too Low (new limits: 9 & 34)
- Computer: Is your # 10?
- User: Too Low (new limits: 11 & 34)
- Computer: Is your # 23?
- User: Too High (new limits: 11 & 22)
- Computer: Is your # 16?
- User: Too Low (new limits: 17 & 22)
- Computer: Is your # 19?
- User: Too High (new limits: 17 & 19)
- Computer: Is your # 17?
- User: Correct
- Computer resets limits to 1 & 100 and asks the user to play again

✏️ 44_SignalSearch

Python

```
#44_SignalSearch.py
"""
    Story: Sasquatch has discovered a strange radio signal deep in the forest,
    but it only transmits sporadically. He needs your help to detect when the
signal stabilizes.

    Instructions:
    Write a program that simulates Sasquatch's signal search.
    Generate a random number between 1 and 20 to represent the signal strength
each time the loop runs.
    The loop should keep running until the signal strength is between 9 and 11
(representing a stable signal).
    Each time the signal is not stable, print a message like "Still searching…"
and the current signal strength.
"""
```

✏️ 45_ElEdMathQuiz

You are writing an elementary math quiz program.

- You start by asking the student how many questions they're going to answer.
    - Normally we'd use a for loop for this, but we're on a while loop unit, so we're going to change it up.
- As long as they haven't been asked that many question yet **and** they haven't gotten 3 questions wrong
    - Generate 2 random #'s between 0 and 9
    - Calculate the answer to the sum of the 2 numbers (answer)
    - Show the user an addition problem with both values and get their response
    - If they're correct, let them know and move on to the next problem.
    - If they're wrong, let them know what the correct answer was and increase the counter for the # of incorrect guesses.
- When done:
    - if they guessed wrong too many times, tell the user to keep practicing.
    - If they didn't guess to many times, tell them they did good and tell them how many incorrect guesses in total they got.

How many problems do you want? 10
6 + 2 = 8
Correct!
1 + 0 = 1
Correct!
7 + 9 = 16
Correct!
5 + 5 = 2
Incorrect - the correct answer was 10
5 + 5 = 6
Incorrect - the correct answer was 10
9 + 3 = 1
Incorrect - the correct answer was 12
Oooo, sorry pal, but you guessed wrong too many times. Practice a bit and come back.

Functions

Functions - Breaking apart code to teach python new tricks.

If python doesn't know a function, we must define it.

def functionName(optionalData):

Why functions?

1. Compartmentalization - package connected content
2. Readability
3. Organization
4. Repeatability (reusability)
5. Adaptability

VendingMachine

- inputs - choice, money
- output - Candy Bar (I expect a valid item back)

Terminology:

- Function Name → The name of a function
- Parameters/Arguments → The values being passed into a function. What a function needs to work
- Return Statement → a line that returns a value back to whatever called the function.
- Call Statement → The command that calls a function
- Function Definition → The section where a function is defined
- Function Signature → Function name + parameters
    - Function signatures must be unique by meeting one of the following criteria
        - Different function name
        - Different number of parameters
    - 2 functions can have the same name if they have a different # of parameters passed into it. (See Function Overloading)
- Function Overloading: When 2 functions have the same name but different # of parameters.

46_Functions

Python

```python
#46_Functions.py
def printRocketTop():
    print(" /\\")
    print("/  \\")
def printRocketMid(repeat):
    for i in range(repeat):
        print("----")
        print("|  |")
        print("----")
def printRocketMid():
    print("----")
    print("|  |")
    print("----")
def printRocketBottom():
    print("\\/\\/")


printRocketTop()
printRocketMid()    #Calls the printRocketMid function with no
parameters/arguments
printRocketBottom()

print()             #Printing empty lines to separate the rockets.
print()
print()

printRocketTop()
printRocketMid(7)   #Calls the printRocketMid function with 1
parameter/argument
printRocketBottom()
```

47_MyMath

Python

```python
#47_MyMath.py

def sum(a, b):
    return num1 + num2
def calcArea(w, h):
    return w * h
#Perimeter Defined:
def calcPerim(w, h):
    return 2 * sum(width, height)


width = int(input("Give me a rectangle's width: "))
height = int(input("Give me a rectangle's height: "))
#result = sum(num1, num2)
#print(f"{num1} + {num2} = {result}")
area = calcArea(width, height)
print(area)
#Now that you have a width and a height, calculate the perimeter
#   Must be done in a function
#   Bonus (kudos) - utilize the sum(#, #) function  → width*2+height*2
#call statement for perimeter:
perimeter = calcPerim(width, height)
```

48_Circle

Python

```python
#48_Circle.py
from math import *
"""
    Ask the user for a radius
    Calculate: (each in their own function)
        1. Diameter → 2*radius
        2. Perimeter → pi * 2*radius
        3. Area → pi * "radius squared"
    print something like:
        "A circle with a radius of 10 has a diameter of 20, a perimeter of
6.28..... and an area of 314.16....."
    *** remember, python has ways to make long decimal values look cleaner.
"""
def calcDiameter(r):
    return 2 * r
def calcPerimeter(r):
    return calcDiameter(r) * pi
def calcArea(r):
    return pi * r * r
radius = int(input("Radius: "))
diameter = calcDiameter(radius)
perimeter = calcPerimeter(radius)
area = calcArea(radius)
print(f"A circle with a radius of {radius} has a diameter of {diameter}, a
perimeter of {perimeter: ,.2f} and an area of {area: ,.2f}")
```

49_OverUnderCooked

Python

```
#49_OverUnderCooked.py
"""
    Define a function called "checkDoneness" that takes in a temperature and
prints whether
    it's safe to eat.

    If the temperature of the meat is less than 165, print "It's Raw!!"
    If the temperature of the meat is greater than 220, print "It's Jerky!!"
    otherwise, print "Very good chef."

    @param temp the temperature of the meat.
"""
```

Allow time for students to try solving on their own

Python

```
#Function def
def checkDoneness(temp):
    if(temp < 165):
        print("It's Raw!!")
    elif(temp > 220):
        print("It's Jerky!!")
    else:
        print("Very Good Chef")
#Main portion of our code
meatTemp = int(input("What is the temperature of the meat?"))
checkDoneness(meatTemp)
"""
    With all programs, you should consider the following:
    1. What is being asked?
       print responses to temp being > or < certain values.
    2. What are the edge cases? (Edge cases are not always apparent)
       Edge Cases: 165, 220
       Should check: edge cases + neighbor #'s (164, 166, 219, 221) + random
numbers
    3. What could go wrong?
       Not much in this program. I guess invalid inputs, but we won't account
for that in this program.
```

```
""
```

✏️ 50_ExactChange

Python

```
#50_ExactChange.py
"""
    Write a function that when given an amount of money (in pennies) it gives
you the lowest amount of
    change possible in quarters, dimes, nickels, and pennies:
        Example: 87 cents → 3 quarter(s), 1 dime(s), 0 nickel(s), 2 penny(s).
    Hints:
        - % finds the remainder when dividing
            87 % 25 → 12
        - // finds the # of times something fits into something else (integer
division)
            87 // 25 → 3
    @param pennies → The total number of pennies provided that you need to
return change for.
"""
#Defining the makeChange function

#Main section of code
pennies = int(input("How many pennies do you have? I'll give you better
change."))
makeChange(pennies)
```

✏️ 51_numSteps

Python

```
#51_numSteps.py
"""
    You are given two integers, A and B. You want to transform A to B by
performing a sequence of
    operations. You can only perform the following operations:
    • Divide A by two, but only if A is even.
    • Add one to A.
    What is the minimum number of operations you need to transform A into B?
    28, 18
    → 14, 18 (divided by 2)
    → 15, 18 (added 1)
    → 16, 18 (added 1)
    → 17, 18 (added 1)
    → 18, 18 (added 1)
    returns 5

    # % 2 == 0 → tells you if something is even
"""
```

✏️ 52_RPS2

Python

```
#52_RPS2.py
#Functions
#def playGame
def playGame():
    print("Playing the game...")
    #write the code that plays a full game of RPS:
        #As long the user hasn't won 2x AND the computer hasn't won 2x
            #get the user's choice (Rock, Paper, or Scissors)
            #get the computer's choice (1, 2, 3)
            #Convert the computer's choice to Rock, Paper, Scissors
            #declare the winner and update the score
"""
    Clarification:
        playgame plays multiple rounds until either the user or the player has
earned 2 points

        *Game: multiple rounds, until 1 person wins 2 rounds.
        *Round: 1 RPS challenge.
"""

#main section
    #Improvements could be made to check for valid responses (yes/no) and
repeat ask if responses are invalid.
play = input("Would you like to play? ('yes' or 'no'... invalid input will be
assumed 'no')")
play = play.lower()
while play == "yes":
    playGame()
    play = input("Would you like to play again? ('yes' or 'no'... invalid input
will be assumed 'no')")
    play = play.lower()
print("Thanks for stopping by? Come play again sometime!")
```

Problem Solving

This unit explores using the tools we've learned so far.

- Operators
    - +, -, *, /
    - //, %, **
    - ()
    - Follows PEMDAS still
        - P: ()
        - E: **
        - MD: *, /, //, %
        - AS: +, -
- Variables
    - Data Types: numbers (integers floating point)
    - Good naming conventions
    - Casting: int(str), float(str)
    - Shortcut operators
        - +=, -=. *=, /=, //=, %=
- Expressions
- Math library (must be imported)
    - pi
    - min & max
    - round, ceil, floor
    - pow, sqrt
-

53_Salad

Python

```
""" 15_Salad.py
    Write a program that calculates how much lettuce, croutons, and dressing is
needed
    for the number of salads being ordered.

    1-2 salads requires 1 head of lettuce.
    1-3 salads requires a bag of croutons.
    1-10 salads requires a bottle of dressing.
    Examples
        1 salads: 1 head(s) of lettuce, 1 bag(s) of croutons, 1 bottle(s) of
dressing
        2 salads: 1 head(s) of lettuce, 1 bag(s) of croutons, 1 bottle(s) of
dressing
        3 salads: 2 head(s) of lettuce, 1 bag(s) of croutons, 1 bottle(s) of
dressing
        4 salads: 2 head(s) of lettuce, 2 bag(s) of croutons, 1 bottle(s) of
dressing
        15 salads: 8 head(s) of lettuce, 5 bag(s) of croutons, 2 bottle(s) of
dressing
    use the ceil(#/#) function from the math library to round up
        15/2 → 7.5
        ceil(15/2) → 8
"""
```

Solution

Python

```
from math import *
numSalads = int(input("How many salads: "))
numLettuce = ceil(numSalads/2)
numCroutons = ceil(numSalads/3)
numDressing = ceil(numSalads/10)
print(f"For {numSalads} salads, you'll need the following:")
print(f"   {numLettuce} heads of lettuce")
print(f"   {numCroutons} bags of croutons")
print(f"   {numDressing} bottle of salad dressing")
```

54_FlipNames

Your task:

- You'll be given a list of space-separated names/words.
- You are to go through finding each name/word, reversing the letters within that name/word, and then printing out the result.
- The result will be the same list of names in the same order, but each individual name is backwards.

Example:

One Two Three Four

becomes

enO owT eerhT ruoF

Solution - searching for words by searching for spaces (manually):

```python
names = "Don Jack Sally Sue Sherlie"
namesReversed = ""
word = ""
for symbol in names:
    if(symbol == " "):
        namesReversed += word[::-1] + " "
        word = ""
    else:
        word += symbol
namesReversed += word[::-1]
print(namesReversed)
```

Solution - searching for words w/ split function

```python
names = "Don Jack Sally Sue Sherlie"
namesReversed = ""
names = names.split(" ")
for name in names:
    namesReversed += name[::-1] + " "
namesReversed = namesReversed.strip()
print(namesReversed)
```

✏️ 55_Billboard (Menu)

Menu/Billboard

***This problem is based off of a problem from the George Fox Programming Competition from 2024(D3, Q7 - Menu).

You're working for a business that has an electronic billboard displaying the item-of-the-day.

You've been frustrated lately because it's your task to manually put the text on the billboard to make it fit just right, but you keep struggling figuring out if the next word will fit and often find yourself redoing work because you misjudged how much space you had.

So, you write a program to do this for you. You have multiple billboards, and they're configured with different widths, so you have to account for that as you write this program.

Your program should get the following inputs from the user:

1. The text that will be displayed on the billboard
2. The number of characters that will fit.

Your program will display the text with a border: dashes (-) across the top, pipes(|) on the side. The width does not include the dashes/pipes.

Preconditions: Your longest word will always be less than or equal to the width.

```
Example:
    input: "Hello it's me and you"
    input: 7
    Output:
        ---------
        |Hello  |
        |it's me|
        |and you|
        ---------
Example:
   input: "Hello it's me and you"
   input: 5
   Output:
       -------
       |Hello|
       |it's |
       |me   |
       |and  |
       |you  |
       -------
```

```
Example:
      input: "Hello it's me and you"
      input: 11
      Output:
          -------------
          |Hello it's |
          |me and you |
          -------------
```

```python
#starter code - this is missing parts, but helps to get text, split it into a list of words, and then loop through each word.
words = input("Provide a message")
words = words.split(" ")
print(words)
for word in words:
    #word represents the current word in your words list
    #len(word) tells you how many characters are in the current word.
```

Additional helper code:

```python
line = "-"*7        #generates "-------"
num = 3
txt = "Hello"
txt += " "*num      #generates "Hello   "
```

This is a hard problem

- Identify what you need to keep track of
- Identify your process (solve this on a white board before you try coding)
- Turn your process into steps
- Turn your steps into code

Hints:

- What do you need to save in this program?
    - List of words (user input turned into a list)
    - Length of billboard (user input turned into a #)
    - Line that will be printed
    - Available space on the line

Pseudocode:

```Python
#Get the user's input and store it into words

#change words to a list of words using the split(" ") function

#Get the billboard's width

#print the dashes


#create a variable to hold onto how much space is left on a line
(availableSpace) and set it equal to the width of the billboard

#create a variable to hold the current line (line) and set it to ""

#For each word in the words list

    #if the length of the current word is larger than the available space

        #line increases by availableSpace spaces

        #reset availableSpace back to the original width of billboard

        #print the line (with | before and after)

        #reset the line to ""

    #add the word to line

    #decrease the size of availableSpace by the length of the word

    #if the availableSpace is greater than 0

        #add a space to line

        #availableSpace decreases by 1

#line increases by availableSpace spaces

#print the line w/ | before and after

#print the dashes
```

✏️ 56_Yoda Speak

(The following is adapted from a George Fox Programming Challenge from 2017, Division II)

I like Star Wars, as do many others. Probably some of you do too if you're into CompSci. If you haven't seen the 7 episodes of the Star Wars saga, that is your homework next weekend. Seriously. You need to see these movies.

Yoda has been one of the iconic characters in the sci-fi realm for many years (since 1981 when The Empire Strikes Back debuted in theaters). Frank Oz, who I was lucky enough to see in Disneyworld many years ago, was the voice of Yoda. His unique speech pattern was part of why we all love Yoda.

In trying to find an algorithm that would describe his speech patterns, here's what I determined. For many of his sentences, it seems that the first two words are put at the end of a sentence. For example, you and I might say, "This is my home." But Yoda would say, "My home this is." Convert a regular English sentence into Yoda-ese.

Input
The first input will contain a single integer n that represents the number of sentences that need to be changed to Yoda Speak. Each sentence will be lowercase with no punctuation. Each sentence will have at least 3 words.
Output
For each data set, Print out each converted sentence.

Example Dialog (`-->` indicates user input)

```
How many sentences will you provide me?
-->4
Give me sentence #1
-->this is my yoda shirt
my yoda shirt this is
Give me sentence #2
-->i am happy to see you
happy to see you i am
Give me sentence #3
-->this is my home
my home this is
Give me sentence #4
-->you are reckless
reckless you are
```

✏️ 57_PathTo1

Path to 1

It's believed (but not proven) that any number can find its way to 1 by following this simple set of steps:

Given a number (from the user)

As long as the # isn't 1:

→ if the number is even, divide by 2.

→ otherwise, multiply by 3 and add 1.

Your task is to show the user the path and tell the user how many even numbers were on your journey to 1 (including the starting number, if it was even).

Example Process (not example dialog):

→ 13

13 (odd: 13*3+1) → 40

40 (even: 40/2) → 20

20 (even: 20/2) → 10

10 (even: 10/2) → 5

5 (odd: 5*3+1) → 16

16 (even: 16/2) → 8

8 (even: 8/2) → 4

4 (even: 4/2) → 2

2 (even: 2/2) → 1

Example Dialog:

What is your starting #? 13

Path: 13-40-20-10-5-16-8-4-2-1

There were 7 even #'s on the path from 13 to 1.

58_Finals Review

Python

```python
#58_Review.py
from math import *
from random import *
#inputs
name = input("What is your name?")
age = int(input("How old are you?"))    #converts String to whole #.
gpa = float(input("What is your gpa?")) #converts String to floating point #.
#print
print(f"You are {name}, your age is {age} and your gpa is {gpa:.2f}.")
print("Hello "+"World") #concatentate Strings
#Operators, variables, and expressions
print(3*4)
print(25/3)     #divides as we expect
print(25 % 3)   #modulate = remainder (left over after dividing)
print(1568427 % 10) #modulate by 10 --> gets last character
print(187 % 2)      #modulate by 2 --> tells even (0) or odd (1 or -1)
print(25//3)    #integer division (cuts off decimal values) - how many 3's fit
in 25
print(1568427 // 10) # integer division by 10 cuts off the last character
num = 3
num = 9 + 7
#num = num + 4
num += 4        # -=, *=, /=, %=, //=
newNum = sqrt(3 * 3 + 4 * 4)    #Pythagorean theorem
print(newNum)
smallNum = min(99, 103, 17)
print(smallNum)
x = sin(3.982)
print(f"{x:.4f}")
#Math functions: Google "w3Schools python math functions"
#booleans and conditionals
brightness = 81
isSunny = brightness > 80
print(isSunny)
if brightness > 80:
    print("It looks so nice and sunny outside!")
else:
    print("It's too cloudy or dark out.")

grade = 75
if grade >= 90:
    print("A")
elif grade >= 80:
```

```
    print("B")
elif grade >= 70:
    print("C")
elif grade >= 60:
    print("D")
else:
    print("F")

#For Loops - Numbered Loops! OR loop through a collection
for i in range(10):         #for i in range(start#: endBefore#: increment)
    print(i)
text = "Hello, I like cheese!"
#loop through each character:
for symbol in text:
    print(symbol)
text = text.split(" ")  #split the text by spaces --> list of words
#loop through each word
for word in text:
    print(word)
#while loops - any non-numbered loop, or going through a partial list
sum = -1
count = 0
while sum != 2 and sum != 12:
    die1 = randint(1, 6)
    die2 = randint(1, 6)
    count += 1
    print(f"Roll #{count}: {die1}, {die2}")
    sum = die1 + die2
#and statements: conditions where both sides must be true to evaluate to true
#or  statements: conditions where at least 1 side must be true to evaluate to
true.
#data validation: looping until a valid response is given
#as long as the user doesn't type ____ and doesn't type ____
resp = input("Play again? Yes, No")
while resp != "Yes" and resp != "No":
    print(f"I don't understand {resp} - just 'Yes' or 'No' please.")
    resp = input()
print(f"Ah, you said "+resp)
#String functions
text = "Hello World"    #every character has an index (address) --> 1st index
is 0
print(text[1])          #Prints 'e'
print(text[4:8])        #Prints 'o Wo'
print(text[8:4:-1])     #Prints 'roW'
```

```
print(text[::-1])       #Prints 'dlroW olleH'
print(text.lower())     #Prints 'hello world'
print(text.upper())     #Prints 'HELLO WORLD'
#Google 'w3schools String functions python' for more useful functions
#Programmer defined Functions
"""
    1. what's its name
    2. what does it need to work?
    3. should it return anything?
"""
#function definition
def pythag(sideA, sideB):
    hyp = sqrt(sideA ** 2 + sideB ** 2)
    return hyp
#call statement
ans = pythag(5, 6)
print(f"The solution to a triangle with sides of 5 and 6 is a hypotenuse of
{ans: .2f}")
```