Welcome to Lecture 15: Intro to Python

1) Open code.cs61a.org
   a) make an account
   b) open a new file
2) Use Iclicker for attendance
3) Lecture 15 Guide: tinyurl.com/S24CS10L15

Agenda: Snap to Python

- Print
- Variables and Types
- User Input
- More Printing (f strings)
- Mathematical Operations
- Functions
- Loops
- Strings Methods

Lecture 15 Guide: tinyurl.com/S24CS10L15

Announcements

- Midterm Retake: Friday July 19 1-4
- For Python:
  - Install Python3
  - Install VS Code
  - *Video Tutorials on ED
    - Go to OH for support
  - If you are using Chromebook, see me after lecture

Lecture 15 Guide: tinyurl.com/S24CS10L15

PREFACE

There will be a significant amount of new content presented in this lecture.

Your job is not to note down every little detail, but instead to try to focus on the differences in Snap! and Python.

You'll internalize most of this content through repeated practice at home, in sections, and in assignments.

COMPUTATIONAL THINKING

CS10 is not a course about Snap! or any particular programming language.

What we're learning is computational thinking:

- How to use computational tools to solve problems.
- Using abstraction to manage complexity.
  - Detail removal
  - Generalization

Lecture 15 Guide: tinyurl.com/S24CS10L15

WHY PYTHON?

- Commonly used language, at Berkeley (CS61A, Data8, etc.) and beyond.
- Per bootcamp.berkeley.edu [1], it's the 2nd most In-Demand programming language.
    - Introducing the programming workflow in a text-based language.
- Open-source libraries, elegant syntax.

- Some things are more easily achieved in Python (eg. list slicing, OOP.)

Lecture 15 Guide: tinyurl.com/S24CS10L15

Task 1

1. Open code.cs61a.org
2. Make an account
3. Create a new file

The `print` statement (similar to “say” block)

```
print(“Hello, world!”)
```

When you want to display text on the screen you use the `print` ***statement.***

The `print` statement

```
print("Hello, world!")
```

Anything you want printed Must be inside parentheses.

The `print` statement

```
print("Hello, world!")
```

Text must be surrounded by quotation marks.

Task 2

1. Open code.cs61a.org
2. Create a new file
3. Type the following:

```
print("Hello, world!")

print("I'm a Python program.")
```

4. Click the green run button

```
Hello, world!

I'm a Python program.
```

Lecture 15 Guide: tinyurl.com/S24CS10L15

Commenting out Code

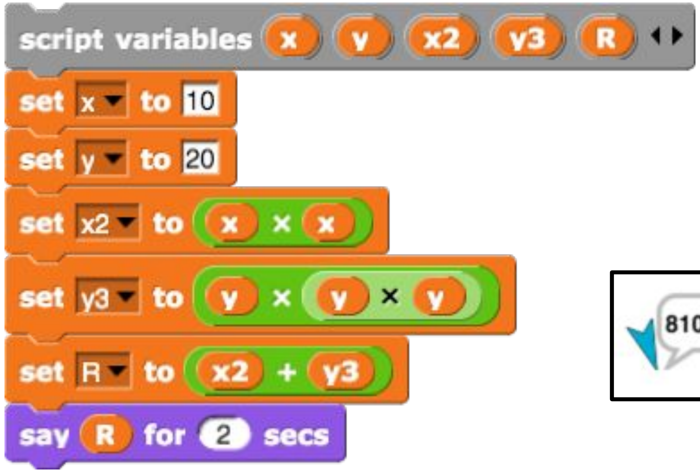
Single Line - use “#” symbol

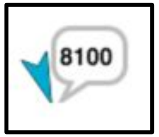
```
# print("Hello, World")
```

Multiple Lines - use “”” “””

```
"""
print("Hello, World")
print("I am a Python Program”)
"""
```

Code Translation: Variables


```
script variables (x) (y) (x2) (y3) (R)
set x to 10
set y to 20
set x2 to (x × x)
set y3 to (y × (y × y))
set R to (x2 + y3)
say R for 2 secs
```



```python
x = 10
y = 20
x2 = x * x
y3 = y * y * y
R = x2 + y3
print(R)
```

- `=` is the assignment operator, similar to `set` in Snap!
- Python evaluates the expression to the right of the operator,
  - assigns the value to the variable (or other location) specified on the left

What is a variable?

A variable has three things:

- name
- type
- value

Meet the variable types: str

- This is short for string.
- A variable whose type is **`str`** contains letters, numbers, punctuation, spaces, etc.
  - Examples: `"Hello,  world! "`, `"abc123”`, `""`

Meet the variable types: int

- This is short for integer.
- A variable whose type is **`int`** contains a number.
- The number can be positive, negative or zero.
- It cannot have a decimal component
    - Examples: `-50, 0, 5`

Meet the variable types: float

- This is short for floating-point number.
- It is similar to an integer, except it ***does*** have a decimal component.
    - Examples: `-3.2, 0.0, 4.5652`

Making a variable

A variable is created and updated in Python using an ***assignment statement***.

```
my_number = 50
```

Making a variable

A variable is created and updated in Python using an ***assignment statement***.

```
my_number = 50
```

variable name

Making a variable

A variable is created and updated in Python using an ***assignment statement***.

```
my_number = 50
```

value

What is a variable?

A variable has three things:

- name: **`greeting`**
- type: **`str`**
- value: **`“Hello, world!”`**

```
greeting = "Hello,  world!"

print(greeting)
```

```
Hello, world!
```

What is a variable?

A variable has three things:

- name: `my_number`
- type: `int`
- value: `50`

```
my_number = 50

print(my_number)
```

```
50
```

Task 3:

Your first variable

```
greeting = "Hello,  world!"

print(greeting)
```

```
Hello, world!
```

Check out this program, which uses a variable to store information.

Your second variable

Here’s another program with a variable.

```
my_number = 50

print(my_number)
```

```
50
```

Finding out type

You can use the `type` functions to determine a variables' type.

It gives you a little more than you need to know, but you can find the type in single quotes after the world `class`.

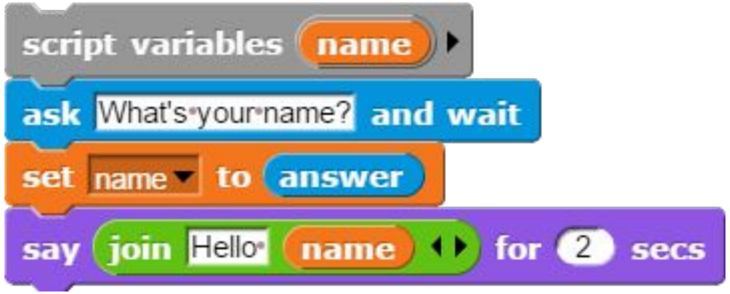
```
greeting = "Hello,  world!"

print(type(greeting))
```

```
<class 'str'>
```

Lecture 15 Guide: tinyurl.com/S24CS10L15

Code Translation: User Input


script variables name
ask What's your name? and wait
set name to answer
say join Hello name for 2 secs

```
name = input("What's your name? ")
print("Hello " + name)
```

Python:

- Assign a variable to user input with input keyword
- In Python, we can join strings by simply adding them with the + sign.

Basic User Input

```
name = input("What is your name: ")
print(name)
```

Basic User Input and Variables

```
name = input("What is your name: ")

print(name)
```

When you want to retrieve text from the user, you use the `input` function.

Basic User Input

```
name = input("What is your name: ")
print(name)
```

This is the prompt that the user sees before they start typing.

Basic User Input

```
name = input("What is your name: ")

print(name)
```

The program pauses on the first line until the user presses the enter key.

Basic User Input

```
name = input("What is your name: ")

print(name)
```

Whatever the user types is then stored in a variable called `name`***.***

Basic User Input

```
name = input("What is your name: ")
print(name)
```

That variable can then be printed like any other variable.

Basic User Input

```
name = input("What is your name: ")

print(name)
```

Here is the shortened version of the program.

```
Name:
```

```
Name: Lisa
Lisa
```

Basic User Input

Here is the shortened version of the program.

What is the type of the variable?

```
name = input("What is your name: ")

print(name)

print(type(name))
```

```
Name: 
```

```
Name: Lisa
Lisa
<class 'str'>
```

Basic User Input

Here is the shortened version of the program.

What is the type of the variable?

```
age = input("Age: ")

print(age)

print(type(age))
```

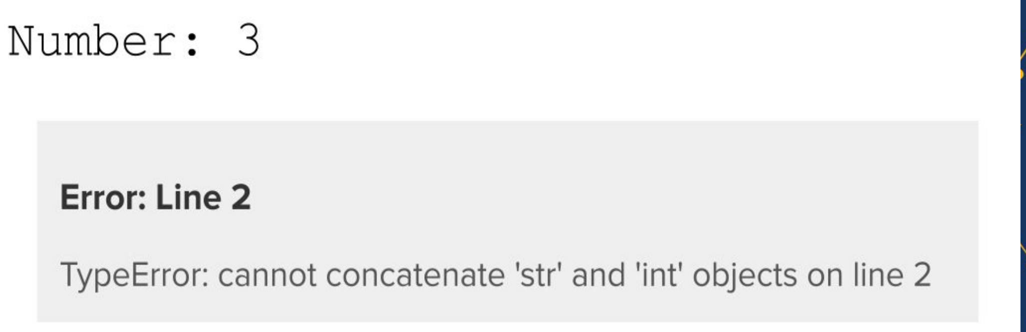
```
Age: 
```

```
Age: 22
22
<class 'str'>
```

Basic User Input

```
number = input("Number: ")
print(number + 1)
```

Why does this cause an error?



```
Number: 3
```

Error: Line 2

TypeError: cannot concatenate 'str' and 'int' objects on line 2

Type Conversion

Let’s take a closer look at this line.

```
number = int(input("Number: "))
```

Type Conversion

Let’s take a closer look at this line.

```
number = int(input("Number: "))
```

First, Python prints the prompt and waits for the user to enter text.

Type Conversion

Let’s take a closer look at this line.

```
number = int(input("Number: "))
```

Then the user’s text is given to the `int` function, which converts it into an integer.

Type Conversion

Let’s take a closer look at this line.

```
number = int(input("Number: "))
```

Finally, the integer is stored in the variable `number`.

Type Conversion

Say the user enters the number 22. Here's what's going on under the hood.

```
number = int(input("Number: "))
```

Type Conversion

Say the user enters the number 22. Here’s what’s going on under the hood.

```
    number = int(input("Number: "))

    number = int("22")
```

Type Conversion

Say the user enters the number 22. Here’s what’s going on under the hood.

```
number = int(input("Number: "))

number = int("22")

number = 22
```

Basic User Input

Fixed!

```python
number = int(input("Number: "))

print(number + 1)
```

```
Number: 
```

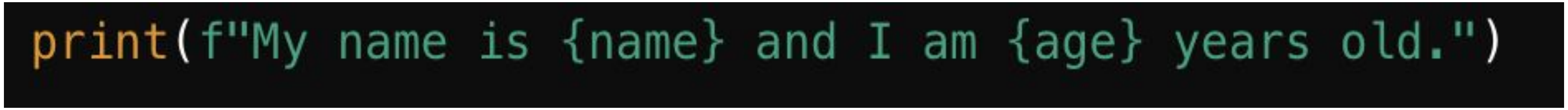
```
Number: 22
23
```

String Interpolation: Using the “+” operator

```
print("My name is " + name + " and I am " + str(age) + " years old.")
```

- Quotation marks around strings
- “+”’s to indicate string variables
- Type conversion to change integer to string

String Interpolation: F Strings



```
print(f"My name is {name} and I am {age} years old.")
```

- Interpolate variables using curly brackets {}
- Don't need to specify variable type

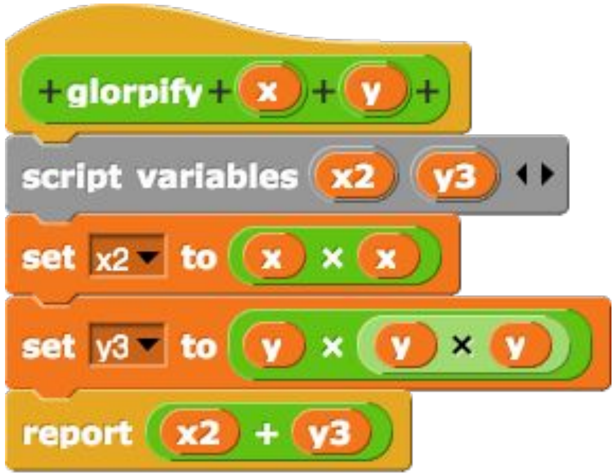
Task 4

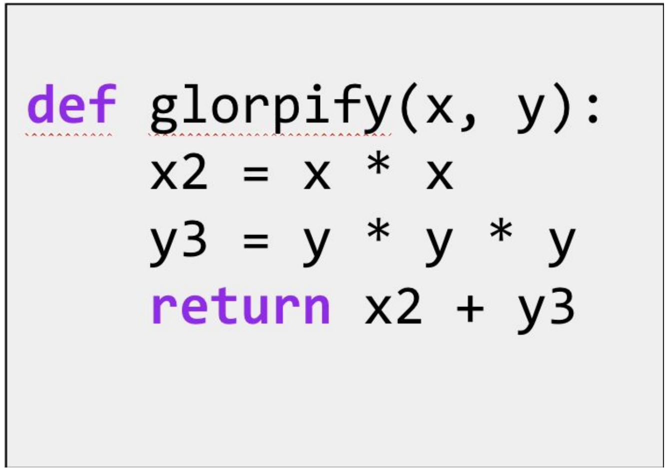
Write a program that:

- Asks the user **their name** and stores in a variable
- Asks the user **their age** and stores in a variable
- Prints out a statement with their name and age
    - Ex: “Professor Rothman is 379 years old”

Lecture 15 Guide: tinyurl.com/S24CS10L15

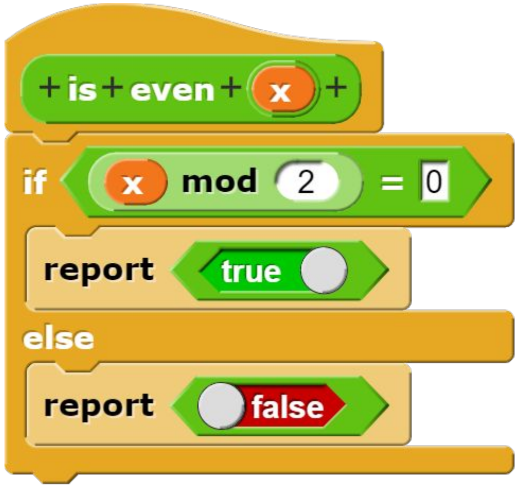
Code Translation: DEFINING A FUNCTION

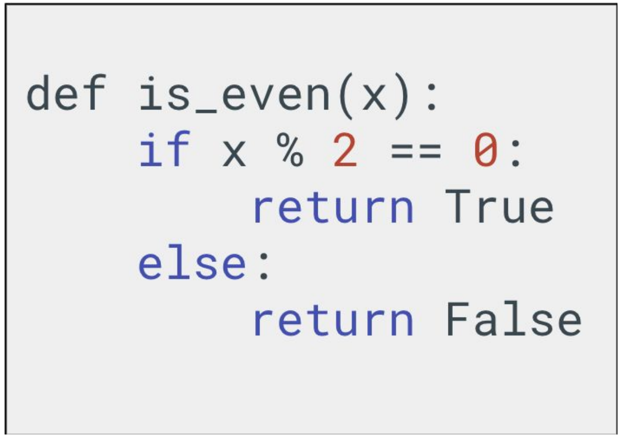

```
+glorpify+ x + y +
script variables x2 y3
set x2 to x × x
set y3 to y × (y × y)
report x2 + y3
```


```
def glorpify(x, y):
    x2 = x * x
    y3 = y * y * y
    return x2 + y3
```

- `def` statements allow us to define functions in Python.
- The body of the function is indented.
- `return` in Python is similar to `report` in Snap!

Code Translation: FUNCTIONS + CONDITIONALS

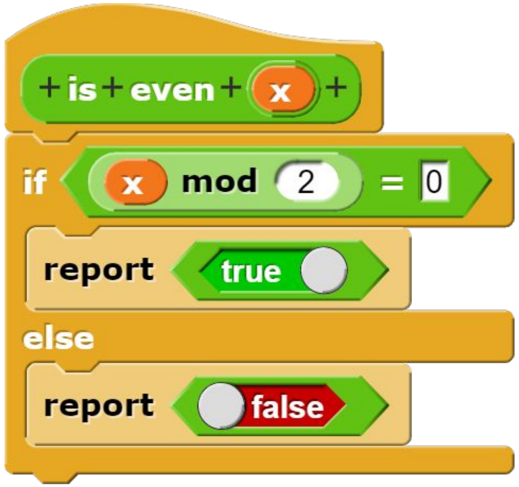




```
def is_even(x):
    if x % 2 == 0:
        return True
    else:
        return False
```

- ==` is the comparison operator, similar to the green predicate `=` in Snap!
- `=` is the assignment operator, similar to the orange command `set` in Snap!

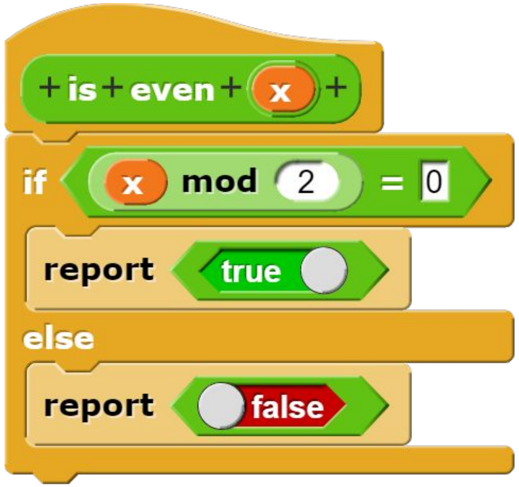
Code Translation: FUNCTIONS + CONDITIONALS



```
def is_even(x):
    if x % 2 == 0:
        return True
    else:
        return False
```

- The boolean values True and False are case-sensitive and capitalized.
- Python `return` == Snap! `report` and Python `print` == Snap! `say`.

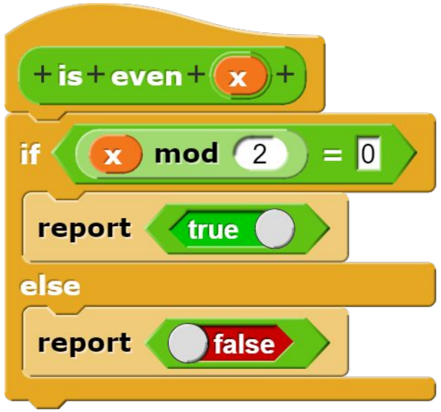
Code Translation: FUNCTIONS + CONDITIONALS



```
def is_even(x):
    if x % 2 == 0:
        return True
    else:
        return False
```

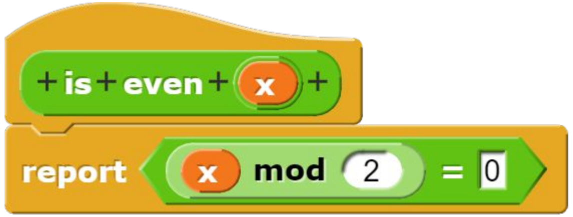
- Observe that we had to indent even further after `if` and `else`.
- Typically, when you see a trailing : (colon), you'll have to indent.

Code Translation: FUNCTIONS + CONDITIONALS



```
+is+even+(x)+
if ((x mod 2) = 0)
    report true
else
    report false
```

```
def is_even(x):
    if x % 2 == 0:
        return True
    else:
        return False
```



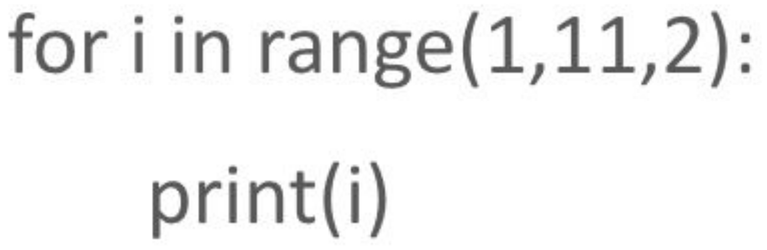
```
+is+even+(x)+
report ((x mod 2) = 0)
```

```
def is_even(x):
    return x % 2 == 0
```

Task: Is this number odd?

- Program a function that takes in an integer
- Returns a Boolean Value (True or False)
- Hint:
  - Use mod (%)

For Loops and Range

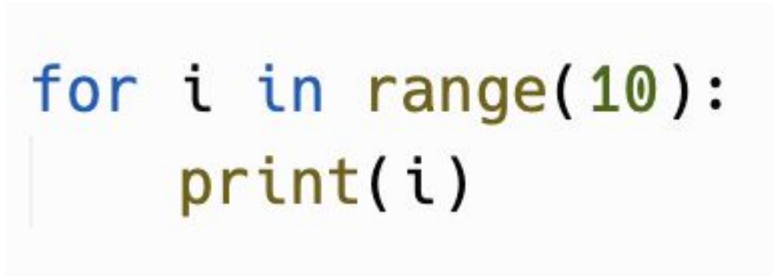


```
for i in range(1,11,2):
    print(i)
```

- starting (inclusive)
- ending (exclusive)
- step (the amount the loop should iterate by…default is 1)
- ***In Python, indexes start at 0!!!**

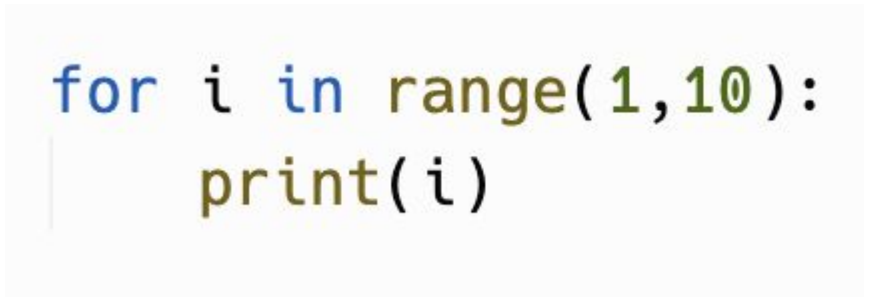
For Loops

This prints ten numbers - 0 through 9.



```python
for i in range(10):
    print(i)
```

This prints nine numbers - 1 through 9.



```python
for i in range(1,10):
    print(i)
```

*1st number is inclusive

*2nd number is excluded

For Loops - How many numbers will this print? - Demo

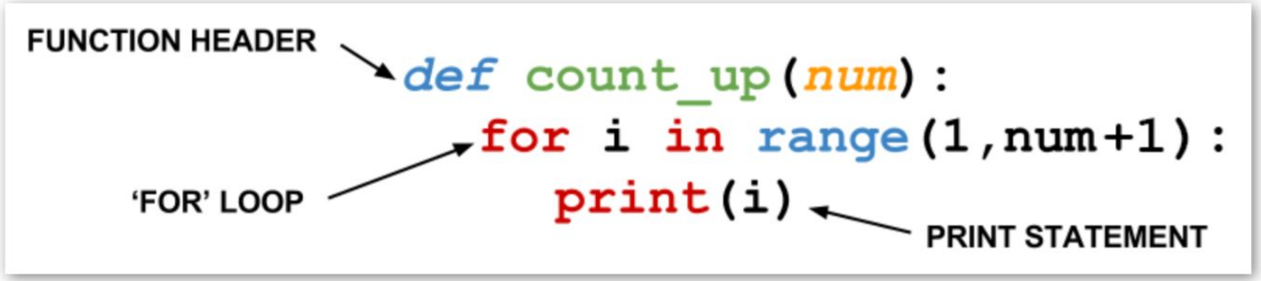
```
for i in range(1,11,2):
    print(i)
```

*1st number is inclusive

*2nd number is excluded

*3rd number is it step iterator

Functions, For Loops and Range

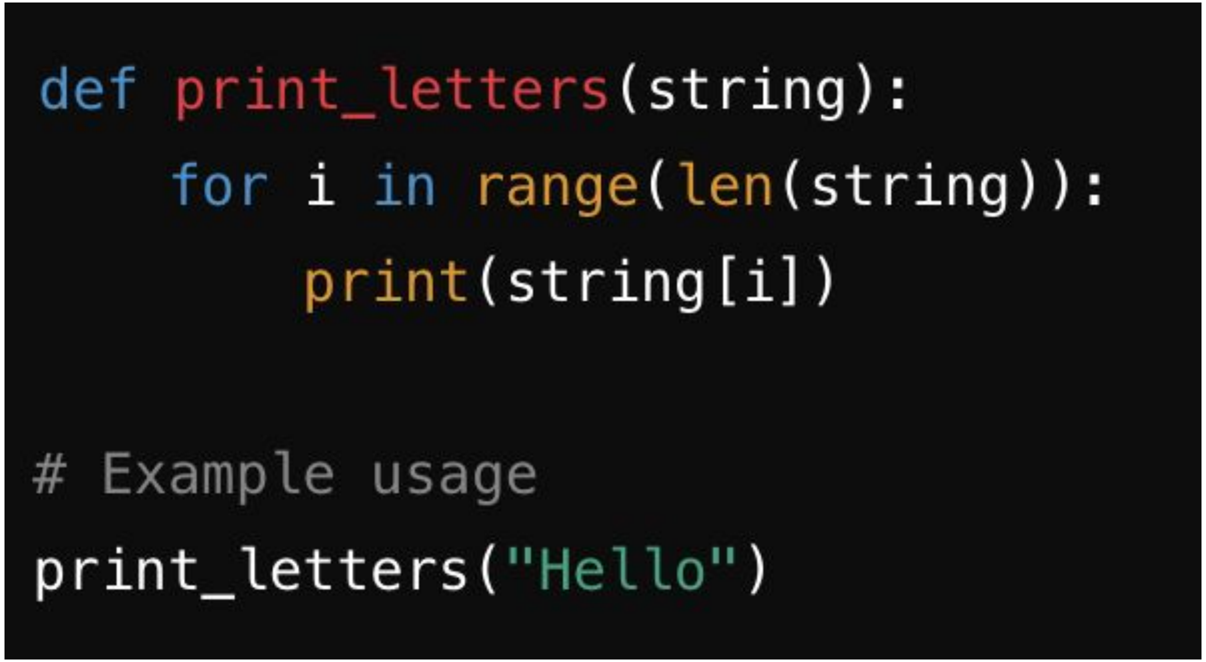


```
def count_up(num):
    for i in range(1,num+1):
        print(i)
```

Task 5: Countdown by 2

- Write a function that takes in a number and counts down (prints) by 2

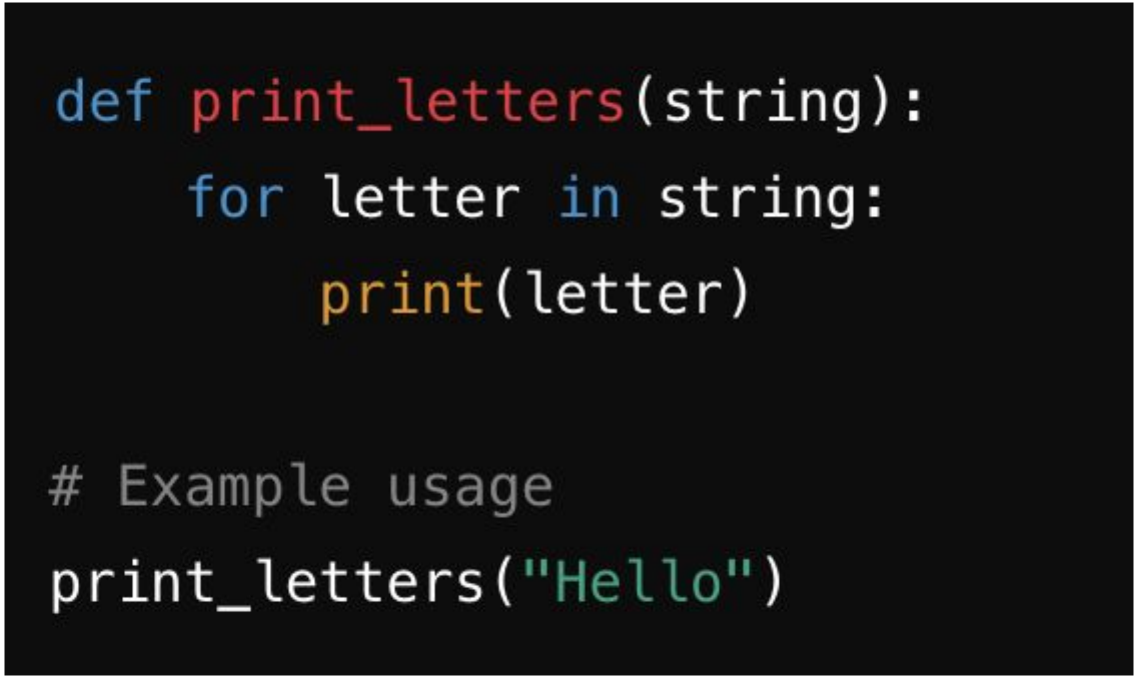
len() function - dynamically evaluates length



```
def print_letters(string):
    for i in range(len(string)):
        print(string[i])

# Example usage
print_letters("Hello")
```

*Using “For Each” Element



```
def print_letters(string):
    for letter in string:
        print(letter)

# Example usage
print_letters("Hello")
```

How can we access the last element in a string?

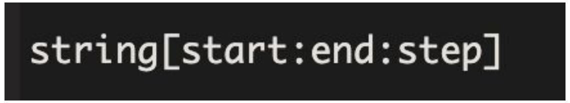
greeting = “Hello!”

- 6 elements
- Index positions 0-5

print(greeting[len(greeting)])

print(greeting[len(greeting)-1])

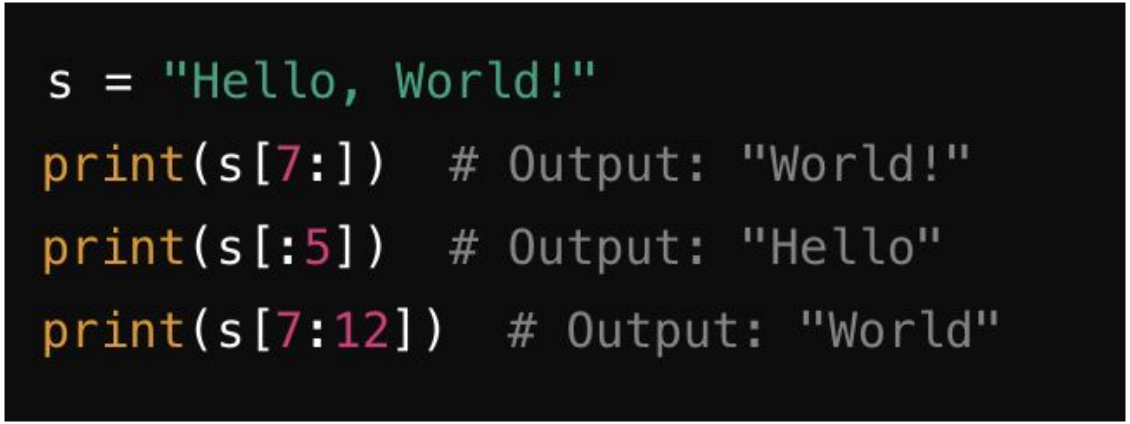
Slice Notation for Strings



```
string[start:end:step]
```

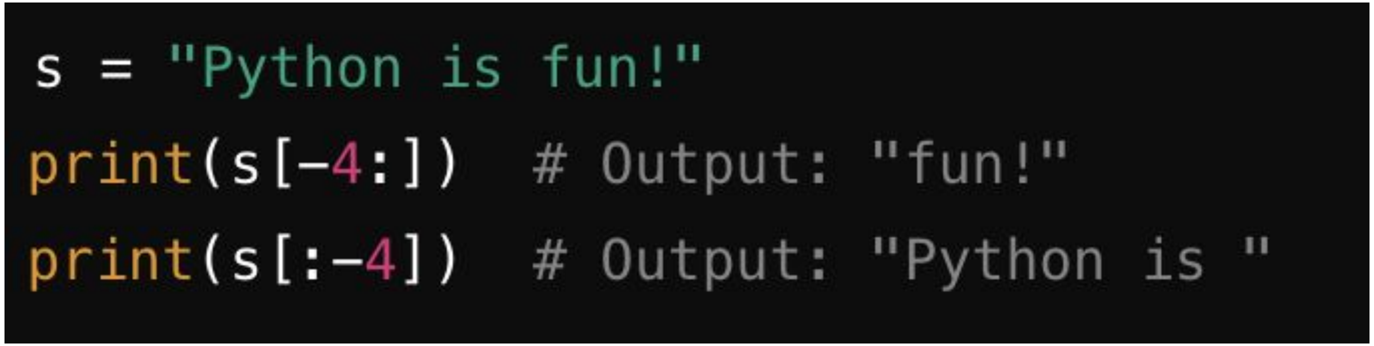
- **start:** Optional parameter indicating the starting index of the slice.
    - If omitted, the slice starts from index 0.
- **end:** Optional parameter indicating the ending index of the slice.
    - The slice will go up to, but not include, the element at this index.
    - If omitted, the slice extends to the end of the string.
- **step:** Optional parameter specifying the step size between elements in the slice.
    - Defaults to 1 if omitted.

Basic Slicing


```
s = "Hello, World!"
print(s[7:])  # Output: "World!"
print(s[:5])  # Output: "Hello"
print(s[7:12])  # Output: "World"
```

- `s[7:]`: Slice from index 7 to the end of the string.
- `s[:5]`: Slice from the start of the string to index 5 (exclusive).
- `s[7:12]`: Slice from index 7 to index 12 (exclusive).

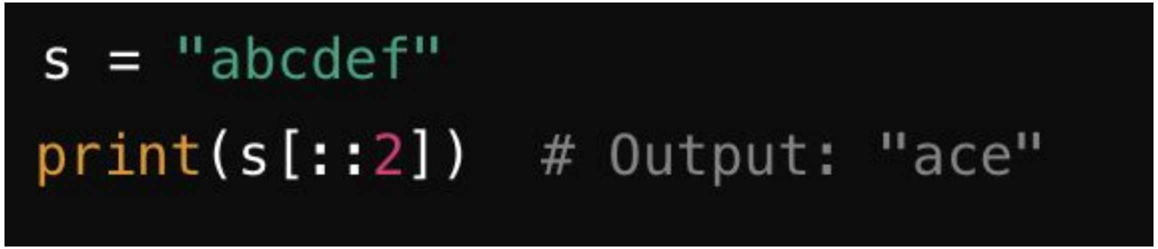
Negative Indices



```
s = "Python is fun!"
print(s[-4:])  # Output: "fun!"
print(s[:-4])  # Output: "Python is "
```

- `s[-4:]`: Slice the last 4 characters of the string.
- `s[:-4]`: Slice from the start of the string up to the 4th last character.

Skipping Characters (Step Iterator)



```python
s = "abcdef"
print(s[::2])  # Output: "ace"
```

- `s[::2]`: Slice the string with a step of 2, i.e., take every second character.

Task 6

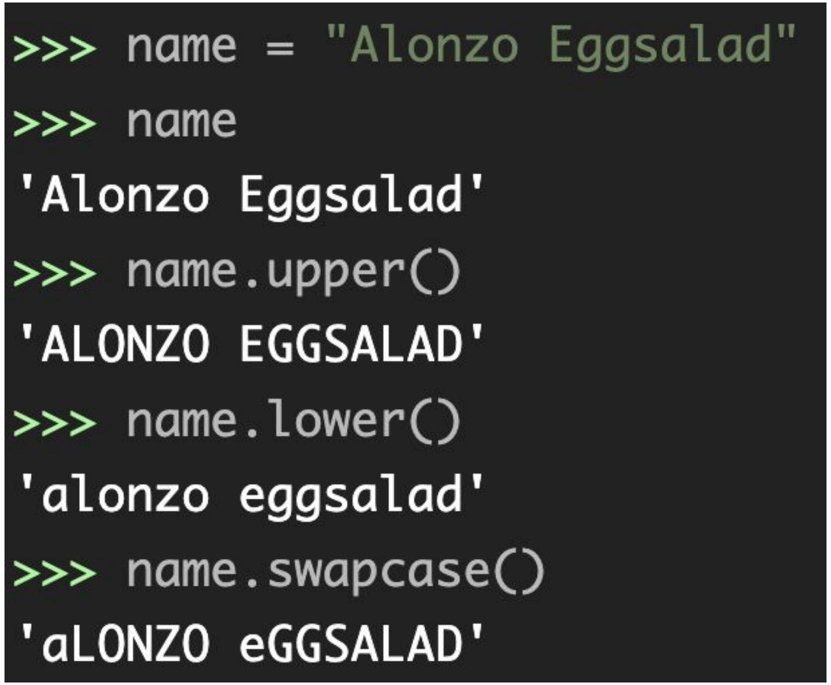
- Write a function that returns a string without 1st and last characters

Lab Exercise #4 Hint

- Google “how to reverse a list with slice in Python”

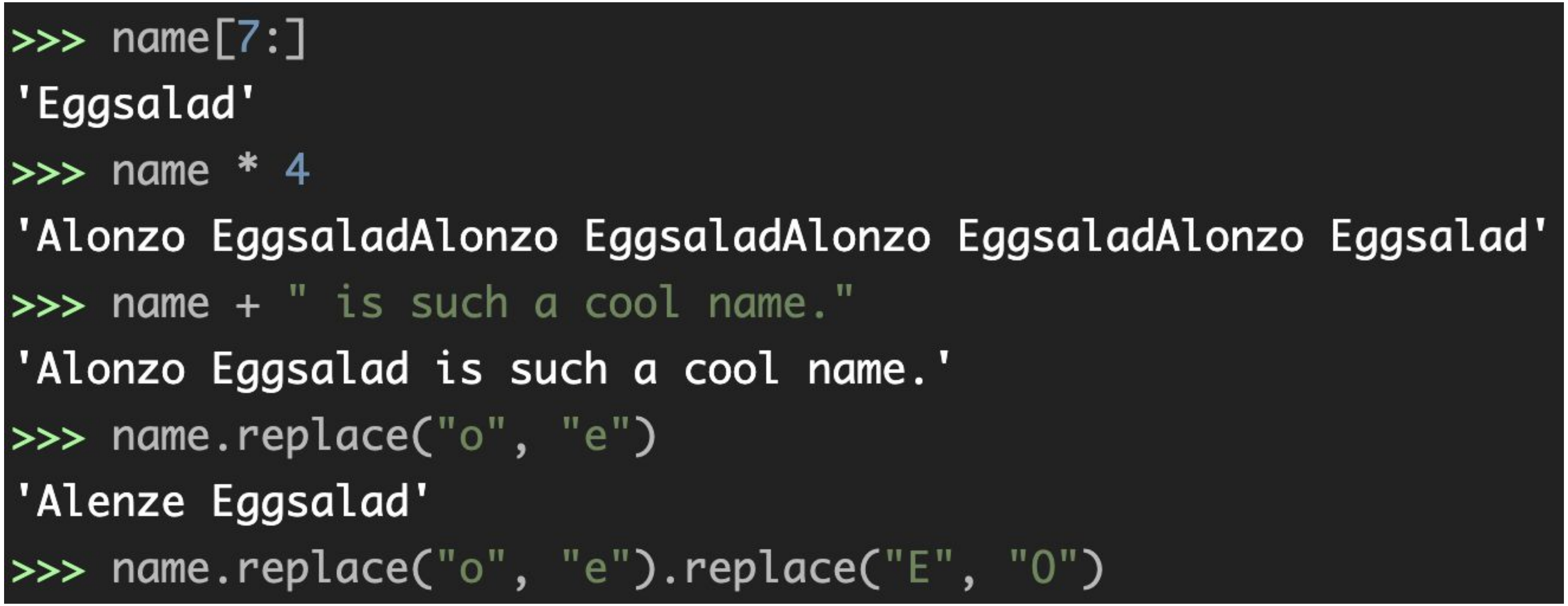
SOME FUN STRING METHODS

A string is a collection of characters.



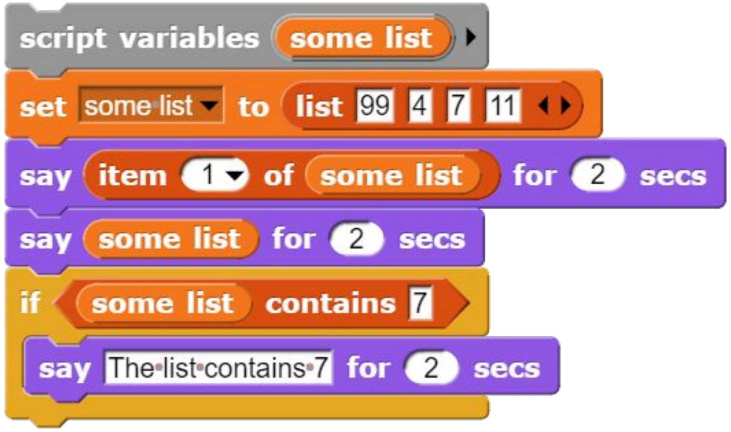
```
>>> name = "Alonzo Eggsalad"
>>> name
'Alonzo Eggsalad'
>>> name.upper()
'ALONZO EGGSALAD'
>>> name.lower()
'alonzo eggsalad'
>>> name.swapcase()
'aLONZO eGGSALAD'
```

SOME FUN STRING METHODS



```
>>> name[7:]
'Eggsalad'
>>> name * 4
'Alonzo EggsaladAlonzo EggsaladAlonzo EggsaladAlonzo Eggsalad'
>>> name + " is such a cool name."
'Alonzo Eggsalad is such a cool name.'
>>> name.replace("o", "e")
'Alenze Eggsalad'
>>> name.replace("o", "e").replace("E", "O")
```

PROGRAM-4: LISTS

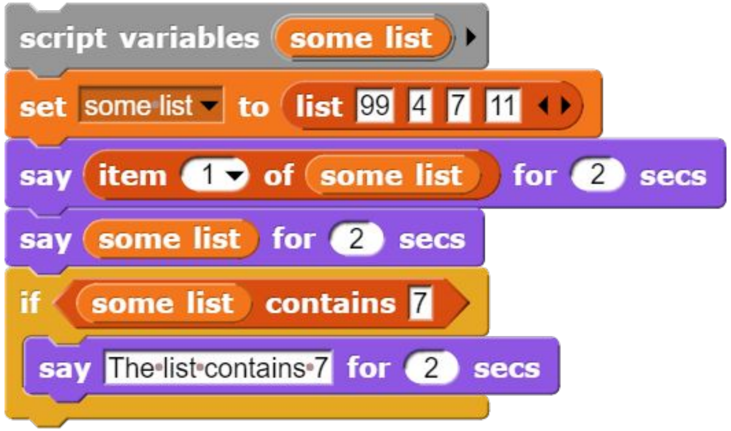


```
script variables (some list)
set [some list] to (list 99 4 7 11)
say (item 1 of (some list)) for 2 secs
say (some list) for 2 secs
if <(some list) contains 7>
    say [The list contains 7] for 2 secs
```

```
some_list = [2, 4, 7, 11]
print(some_list[0])
print(some_list)
if 7 in some_list:
    print("The list contains 7")
```

Square-bracket notation for lists!

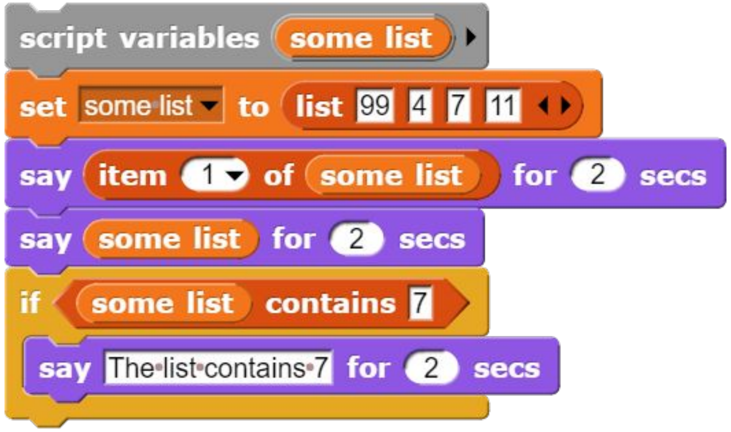
PROGRAM-4: LISTS


script variables some list
set some list to list 99 4 7 11
say item 1 of some list for 2 secs
say some list for 2 secs
if some list contains 7
say The list contains 7 for 2 secs

```python
some_list = [2, 4, 7, 11]
print(some_list[0])
print(some_list)
if 7 in some_list:
    print("The list contains 7")
```

`some_list[0]` is similar to `item 1 of some_list`
Square-brackets are used to index into lists (retrieve items at given positions.)
Furthermore, Python is 0-indexed, so the first item lives in index 0.

PROGRAM-4: LISTS


script variables some list
set some list to list 99 4 7 11
say item 1 of some list for 2 secs
say some list for 2 secs
if some list contains 7
say The list contains 7 for 2 secs

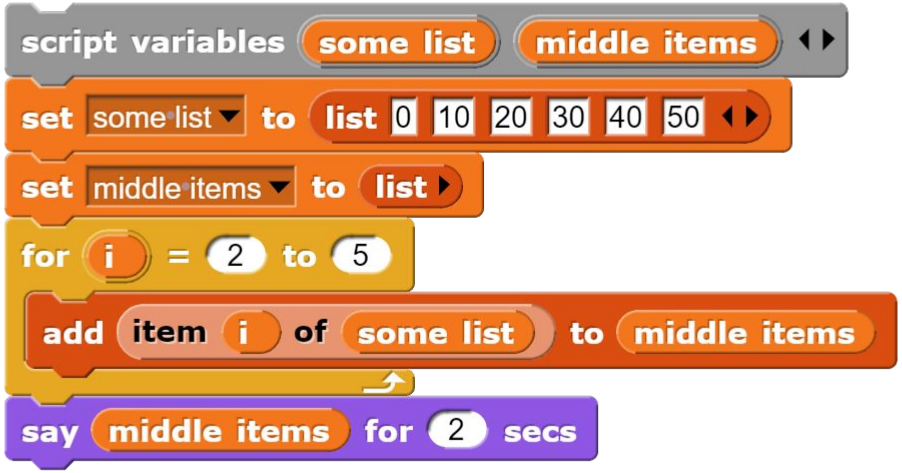
```
some_list = [2, 4, 7, 11]
print(some_list[0])
print(some_list)
if 7 in some_list:
    print("The list contains 7")
```

`in` is the membership operator in Python, similar to `contains`.

CREDIT: LOTS OF CONTENT BORROWED FROM JOSH HUG'S MATERIAL.

TRICKIER CODE TRANSLATION

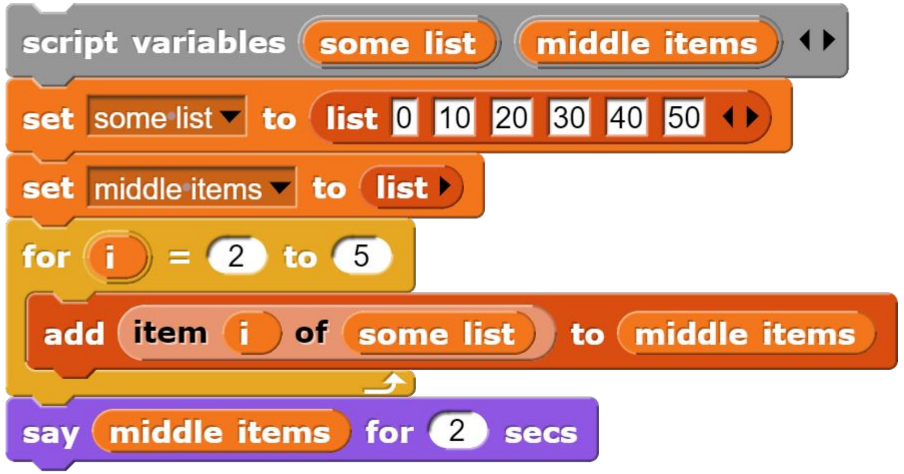
PROGRAM-5: MIDDLE ITEMS


```
script variables (some list) (middle items)
set some list to list 0 10 20 30 40 50
set middle items to list
for i = 2 to 5
    add item i of some list to middle items
say middle items for 2 secs
```

```
some_list = [0, 10, 20, 30, 40, 50]
middle_items = some_list[1:5]
print(middle_items)
```

Python code seems simpler / more elegant.

PROGRAM-5: MIDDLE ITEMS


```
script variables (some list) (middle items)
set some list to (list 0 10 20 30 40 50)
set middle items to (list)
for i = 2 to 5
    add (item i of some list) to middle items
say middle items for 2 secs
```

```
some_list = [0, 10, 20, 30, 40, 50]
middle_items = some_list[1:5]
print(middle_items)
```

SLICE NOTATION

slices

```
L = [0, 10, 20, 30, 40, 50]
```

SLICE NOTATION

Allows us to access slices of lists.

```
L = [0, 10, 20, 30, 40, 50]
```

```
L[1:5]    → [10, 20, 30, 40]
L[2:]     → [20, 30, 40, 50]
L[:3]     → [0, 10, 20]
```

7