Python (Week 1)

A high-level, open-source, purely Object-Oriented, general programming language

Cognixia™

Outline

Prerequisites

1. https://www.python.org/downloads/
   - Download Python for your Operating System
2. https://code.visualstudio.com/
   - Visual Studio Code is the current standard for Integrated Development Environments
   - The **Python** and **Pylance** extensions are recommended
3. https://www.anaconda.com/products/individual
   - Data focused distribution
   - The Anaconda distribution provides a suite of tools for data science

Python Installation

Installation

- Windows
  - When installing via the download, be sure to add Python to the **environment variables**
  - Python can be installed via Chocolatey `choco install python`
  - A development only edition of Python is also available through the Windows store
- Mac OS
  - OS X comes with **Python 2.7** installed, you must install **Python 3**
  - Python 3 can also be installed through homebrew with the command `brew install python3`
- Linux
  - Many distributions of of Linux come with Python 3 already installed
  - The command `sudo apt-get install python3.6` can be used to install a specific version

The Python Language

Python History

- Python’s first deployment was in 1990
- Developed by the Python Software Foundation
  - Centrum Wiskunde & Informatica (National Research Institute for Mathematics and Computer Science) in the Netherlands
- The principal author of the language was Guido van Rossum
- Python 2.0 was released in 2000
- Python 3.0 was released in 2008
  - It was not backwards compatible with Python 2

Python Attributes

1. Open Source
2. High Level
3. Interpreted
4. Object Oriented
5. Multi-Paradigm
6. Extensible

The Zen of Python

1: Beautiful is better than ugly	2: Explicit is better than implicit
3: Simple is better than complex	4: Complex is better than complicated
5: Flat is better than nested	6: Sparse is better than dense.
7: Readability counts	8: Special cases aren't special enough to break the rules
9: Although practicality beats purity	10: Errors should never pass silently
11: Unless explicitly silenced	12: In the face of ambiguity, refuse the temptation to guess
13: There should be one-- and preferably only one --obvious way to do it	14: Although that way may not be obvious at first unless you're Dutch
15: Now is better than never	16: Although never is often better than *right* now
17: If the implementation is hard to explain, it's a bad idea	18: If the implementation is easy to explain, it may be a good idea
19: Namespaces are one honking great idea -- let's do more of those!	20:

Python Pros and Cons

Advantages

- ➔ Easy to learn; naturalistic syntax
- ➔ Powerful; extensible with libraries
- ➔ Popular with an active community

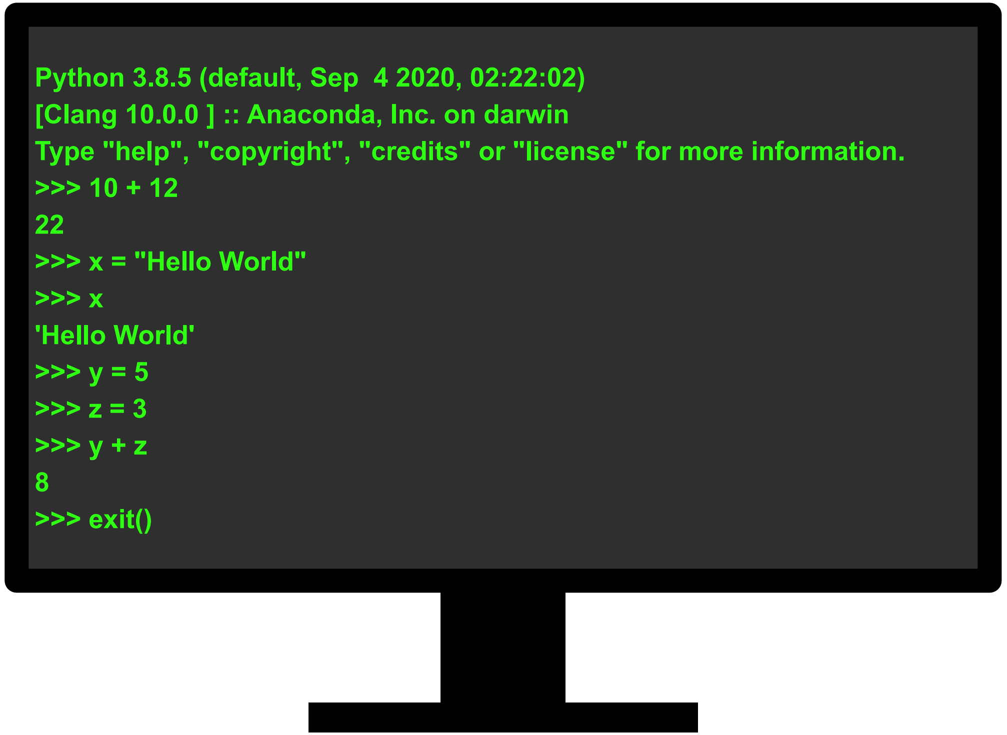
Disadvantages

- ➔ Interpreted, not Compiled
- ➔ Comparatively high memory usage
- ➔ Dynamic Typing

Fundamentals and Syntax

Python Shell

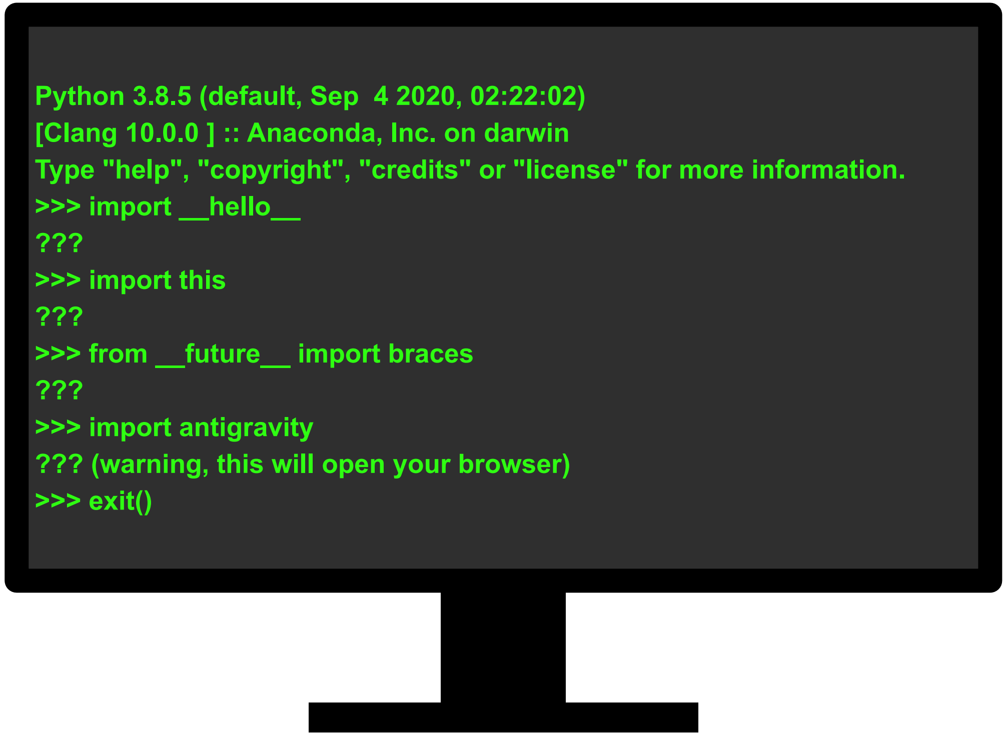
- ➔ To check your installation, enter one of the following commands:
  - ◆ `python --version`
  - ◆ `python3 --version`
- ➔ Enter the following command to enter the Python Shell
  - ◆ `python`
  - ◆ `python3`
- ➔ Enter commands directly into the Command lIne
- ➔ The following command will exit
  - ◆ `exit()`



```
Python 3.8.5 (default, Sep  4 2020, 02:22:02)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 10 + 12
22
>>> x = "Hello World"
>>> x
'Hello World'
>>> y = 5
>>> z = 3
>>> y + z
8
>>> exit()
```

Easter Eggs and Jokes

- ➔ Python was named after **Monty Python's Flying Circus**
  - ◆ Non-essential parts of the code contain references to several pop culture properties
- ➔ Here are some potentially amusing shell commands to try:
  - ◆ import __hello__
  - ◆ import this
  - ◆ from __future__ import braces
  - ◆ import antigravity
    - ● (minor warning, this one will open your browser)



```
Python 3.8.5 (default, Sep  4 2020, 02:22:02)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import __hello__
???
>>> import this
???
>>> from __future__ import braces
???
>>> import antigravity
??? (warning, this will open your browser)
>>> exit()
```

.py Files

- ➔ Most Python development is done through modifying **.py** files
- ➔ Python code is executed sequentially, one instruction at a time
  - ◆ Functions are not hoisted
- ➔ A .py file can be executed from the **terminal** with
  - ◆ `python <filename>.py`
- ➔ **Jupyter Notebook** cells can be executed individually
- ➔ The **Spyder** environment allows you to execute specific lines

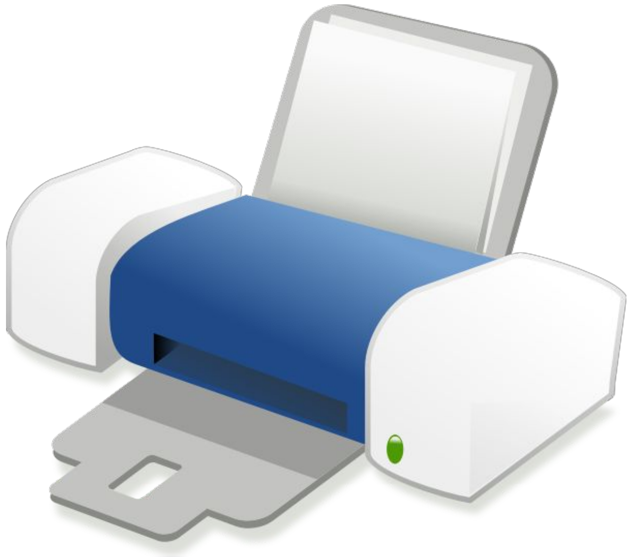
Comments

➔ Python Single line comments are declared with #

```
# Well commented code is essential any project
```

➔ Python does not officially support multi-line comments, but any strings not assigned to a variable will be skipped by the interpreter

```
"This line will be skipped by the interpreter"
"""
The triple quotation mark syntax starts a multi-line string
   They will preserve both indentation and line spacing
This is standard syntax for multi-line comments in Python
"""
```



The Print Function

- ➔ The print function will print a value to standard output
  - ◆ It is a globally available function

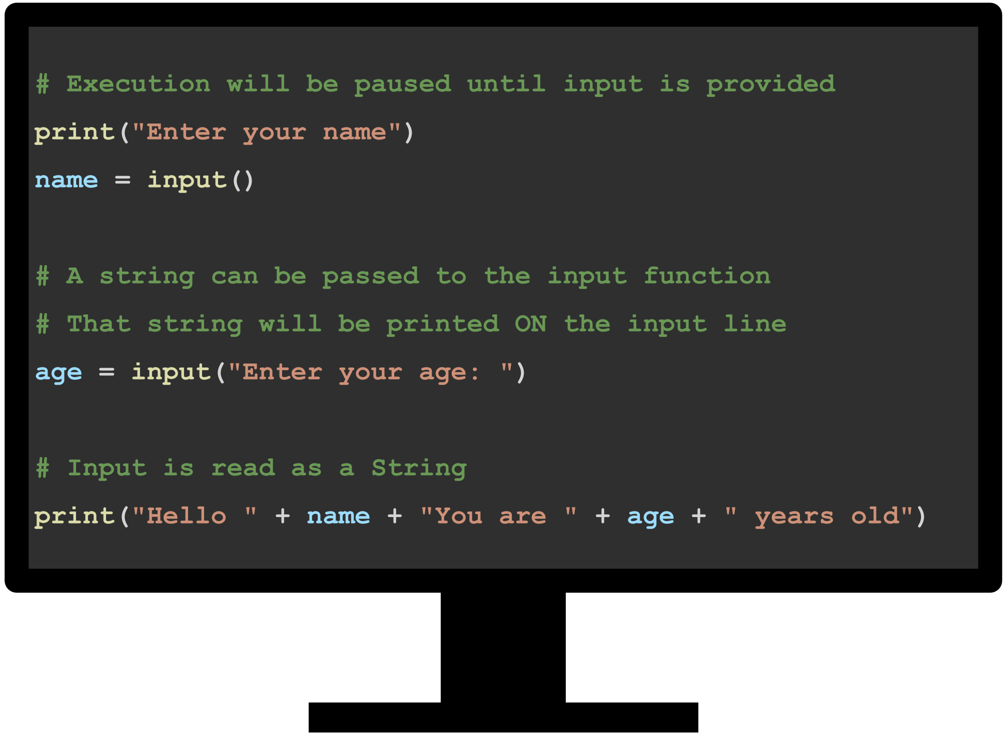
```
print("Hello World")
```

- ➔ Print can accept any data type
  - ◆ Data Types cannot be directly mixed

```
print (["Hello", "World"])
print (42)
# Will not work.
# print ("Hello World" + 42)
```

Input Function

- ➔ The built-in input function will read from the standard input
- ➔ Characters are read as Strings
- ➔ The execution of the program will stop until input is entered



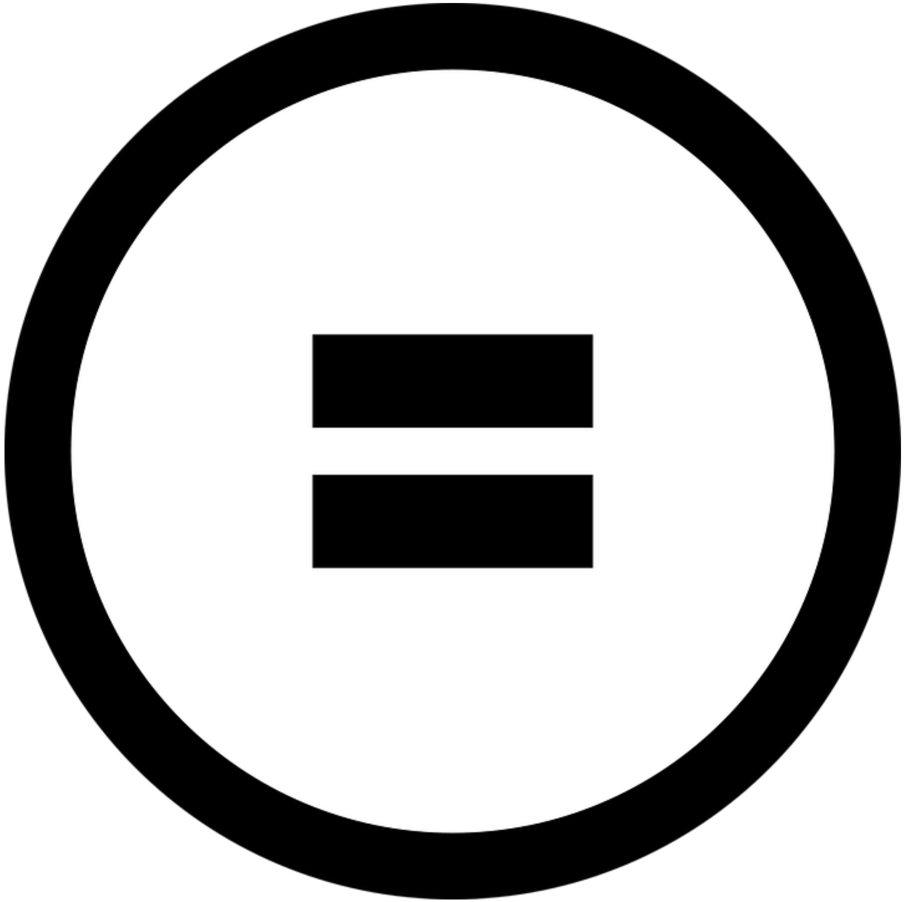
```
# Execution will be paused until input is provided
print("Enter your name")
name = input()

# A string can be passed to the input function
# That string will be printed ON the input line
age = input("Enter your age: ")

# Input is read as a String
print("Hello " + name + "You are " + age + " years old")
```

Reserved Words

False	None	True	and	as
assert	async	await	break	class
continue	def	del	elif	else
except	finally	for	from	global
if	import	is	in	lambda
nonlocal	not	or	pass	raise
return	try	while	with	yield

Student Exercise

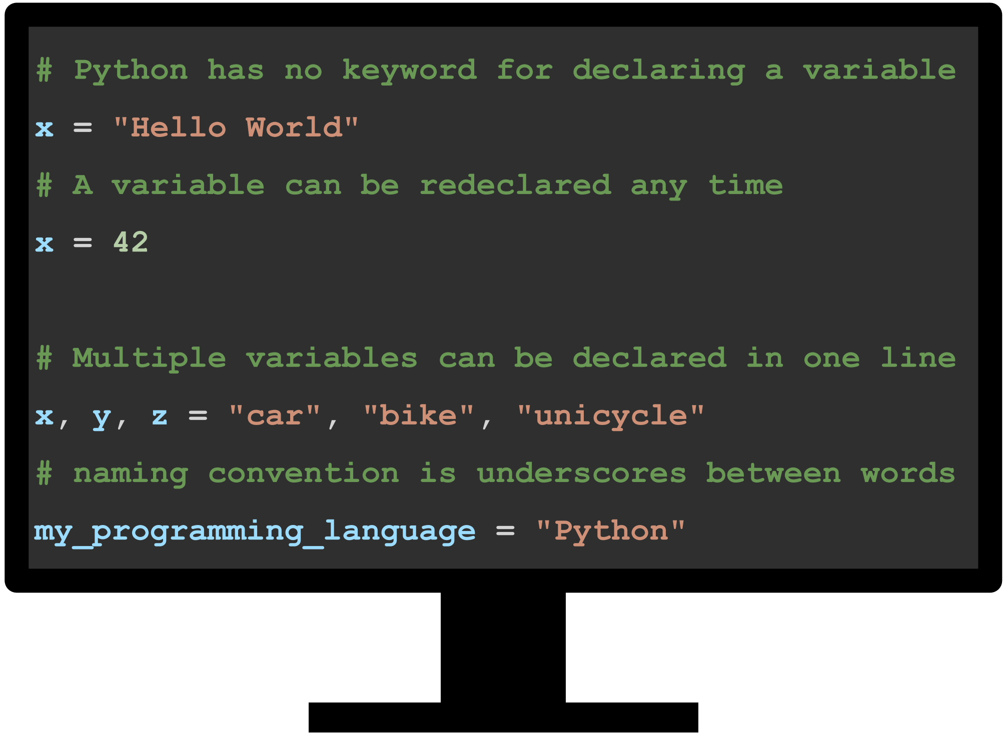
- ➔ Let's make sure all of the installations worked properly!
- ➔ Create a .py file that accepts user input and formats an output
  - ◆ **Prompt for the user for their name**
  - ◆ **Prompt the user for their age**
  - ◆ **Prompt the user for their profession**
  - ◆ **Print all of the information entered by the user in a single sentence**



Variables and Data Types

Dynamic Typing

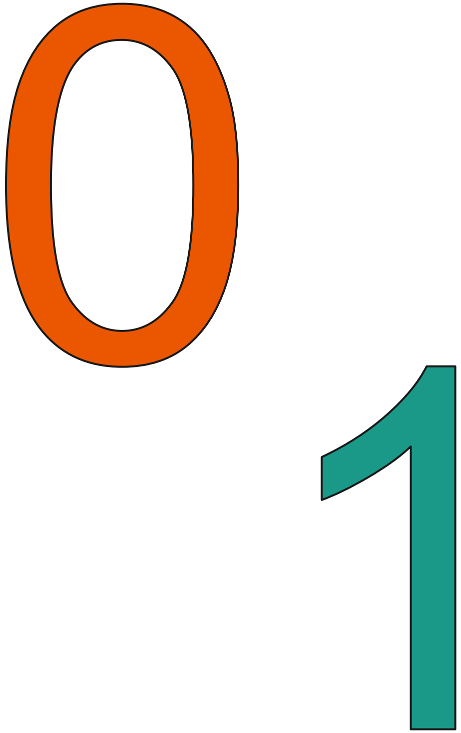
- Python is a **Dynamically-typed** language
  - The type of variable is not determined on **declaration**
  - A variable can be redeclared at any time to **any type**
  - Functions have no fixed return type or argument types
- There are extensions to Python to add static type checking
  - mypy



```python
# Python has no keyword for declaring a variable
x = "Hello World"
# A variable can be redeclared any time
x = 42

# Multiple variables can be declared in one line
x, y, z = "car", "bike", "unicycle"
# naming convention is underscores between words
my_programming_language = "Python"
```

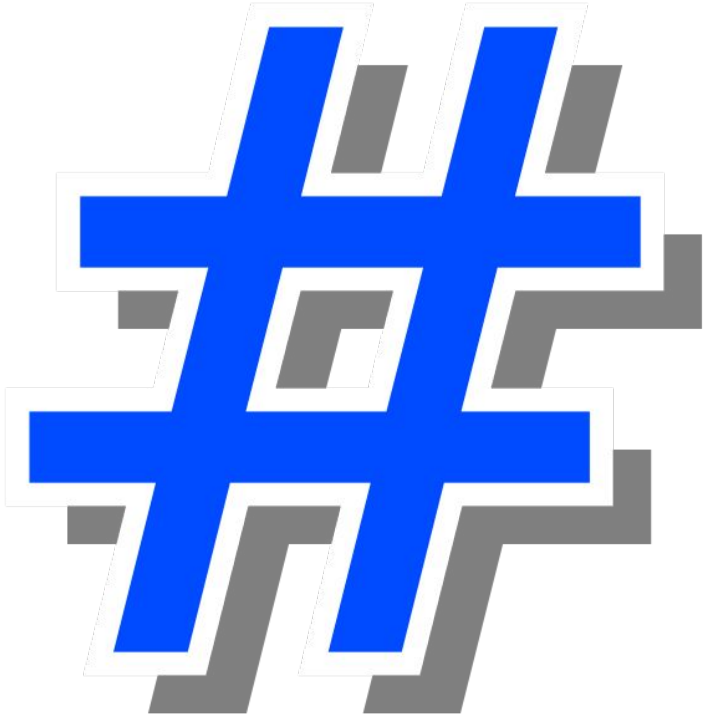
Booleans

- Identified by **bool**
- Have a value of True or False
  - Boolean variables are capitalized in Python
- Logical operators resolve to Booleans



Booleans

- ➔ Objects with content resolve to **True**:
  - The keyword **True**
  - Any number other than **0**
  - Any String other than the empty String
  - tuples (<items>), lists [<items>] , and dicts or sets {<items> } with content
- ➔ The following values resolve to **False**:
  - The keywords **None** and **False**
  - The number **0**
  - Empty strings “”
  - Empty tuples ( ), lists [ ] , and dicts or sets { },



Numbers

- Python has 3 number types:
  - **int**: Whole numbers, positive or negative, of arbitrary length

```
x = 10
```

  - **float**: Positive or negative number containing one or more decimals

```
x = 37.43
```

  - **complex**: Numbers with an imaginary component, denoted with a j

```
x = 6j
```

- Python has a robust collection of mathematical features

Strings

- ➔ Identified by **str**
- ➔ Python considers strings as arrays of unicode characters
    - ◆ Strings support all List methods, as well as their own methods
- ➔ Python strings can be declared with single or double quotes

```python
first_string = 'Hello'
other_string = "World"
```

- ➔ **len()**
    - ◆ Globally available function
    - ◆ Returns the length of a string

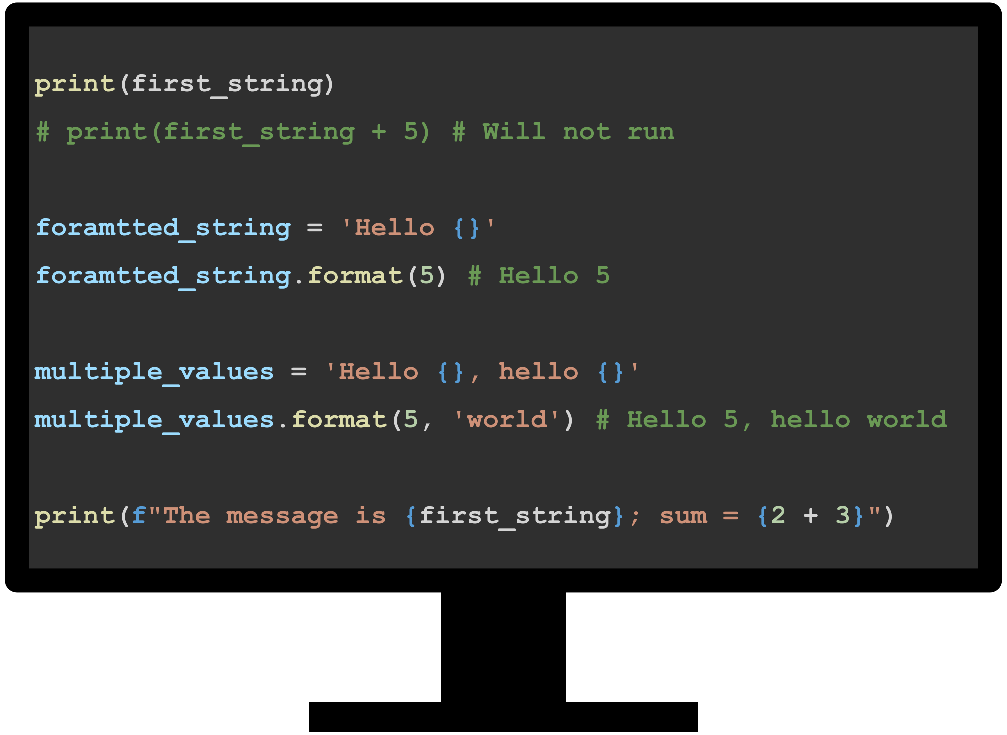
```python
len("Hello World") # returns 11
```

String Methods

Method	Description
.capitalize()	Converts the first character to upper case
.count(str)	Returns the number of times a specified value occurs in a string
.find(str)	Searches the string for a specified value and returns the position of where it was found, returns -1 if not
.index(str)	Same functionality as .find() except it raises an exception if the substring is not found
.join(iterable)	Joins the elements of the passed iterable with the string
.replace(substr, str)	Returns a string where a specified value is replaced with a specified value
.split(substr)	Splits the string at the specified separator, and returns a list
.upper()	Converts a string into upper case

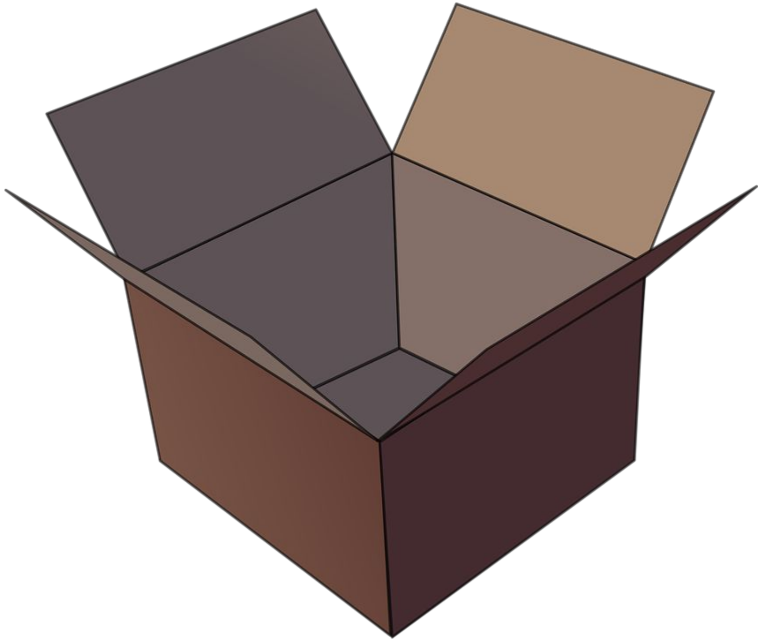
String Format

- ➔ **Strings** can be concatenated to other **Strings**, but not to other data types
- ➔ The The **.format()** method will add any character to a string in the curly braces
  - ◆ Multiple values can be added to a string
- ➔ The **f string** syntax automatically formats strings



```
print(first_string)
# print(first_string + 5) # Will not run

foramtted_string = 'Hello {}'
foramtted_string.format(5) # Hello 5

multiple_values = 'Hello {}, hello {}'
multiple_values.format(5, 'world') # Hello 5, hello world

print(f"The message is {first_string}; sum = {2 + 3}")
```

None



- ➔ **None** represents an empty value
  - ◆ This is different from 0, or the empty string
- ➔ **None** is an object, but has no methods
- ➔ All **None** values share the same object

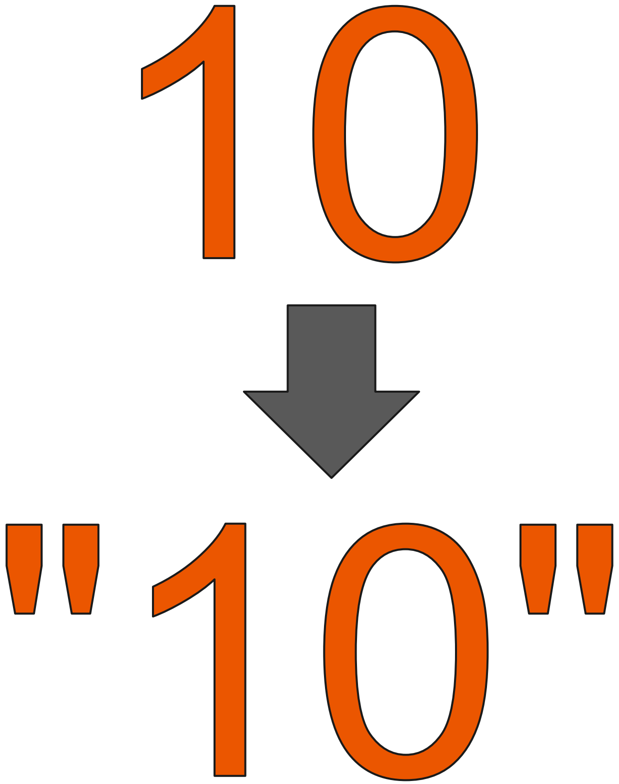
```
empty = None
```

Checking Data Types

- ➔ type(variable)
  - ◆ Globally available function
  - ◆ Returns the class of the type of object
  - ◆ str, int, float, complex, bool
- ➔ isinstance(variable, class)
  - ◆ Globally available function
  - ◆ Returns a boolean based on if the passed in variable is an instance of the given class

Casting Data Types

- **int()**
  - converts a string to an int
  - floors a float
- **float()**
  - add a .0 to a int
  - converts a string to a float
- **str()**
  - can take in a variety of arguments and converts them to a string
- If a variable is attempted to be cast to an incompatible data type, a **ValueError** is thrown



del

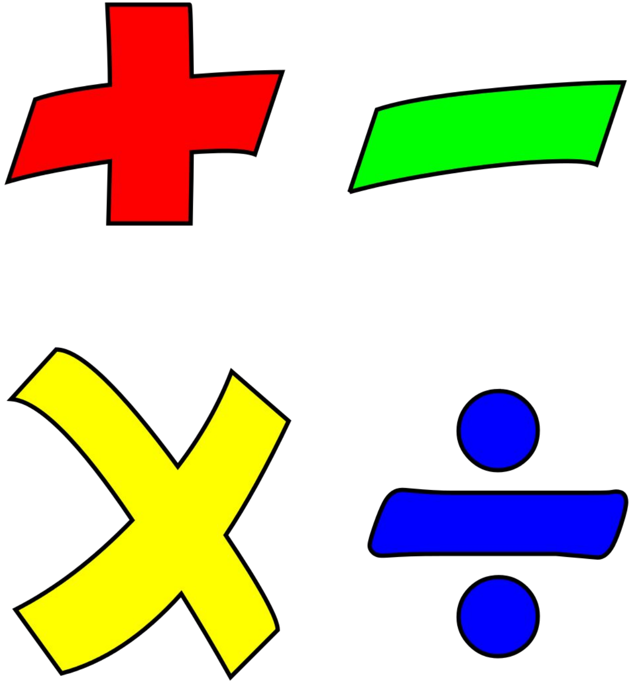
- → The **del** keyword will remove a variable from the list of names
- → Unreferenced objects are removed from memory

```
x = 12
print(x) # 12
del x
print(x) # NameError: name 'x' is not defined
```

Student Exercise

- ➔ Write a program to capture user input for an Employee
- ➔ Record the employee's name
  - ◆ Split the name into first and last
  - ◆ Make sure that the first letter of both is capitalized, and the rest of the letters are lowercase
- ➔ Record the employee's age
  - ◆ Parse the age information to an int
  - ◆ Record the employee's birth year
- ➔ Generate the employee's email
  - ◆ Concatenate the first and last names with a "."
  - ◆ Add the last two digits of their birth year to the last name
  - ◆ Add @company.com to the end
- ➔ Print all results to the screen

Python Operators



Arithmetic Operators

Operator	Name	Example
+	Addition	9 + 5 # 14
-	Subtraction	8 - 3 # 5
*	Multiplication	5 * 7 # 35
/	Division	4 / 2 # 2.0
%	Modulus	7 % 3 # 1
**	Exponential	5 ** 3 # 125
//	Floor Division	9 // 4 # 2

Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero Fill Left Shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed Right Shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Assignment Operators

Operator	Name	Example
`=`	Assignment	x = 42 # 42

`+=`, `-=`, `*=`, `/=`, `%=`, `**=`, `//=`	Performs a mathematical operation then assigns a variable	y = 10
		y += 7
		# 17

`&=`, `\=`, `^=`, `>>=`, `<<=`	Performs a bitwise operation then assigns the variable	z = 5 z &= 3 # 1

Comparison Operators

Operator	Name	Example of True
==	Equal To	10 == 10
!=	Not Equals	10 != 11
>	Greater Than	10 > 9
<	Less Than	10 < 20
>=	Greater Than or Equal To	10 >= 10
<=	Less Than or Equal To	10 <= 10

Identity Operators

Operator	Description	Example of True
`is`	Returns True if both variables are the same object	“apple” `is` “apple”
`is not`	Returns True if both variables are not the same object	“apple” `is not` [1,2,3]

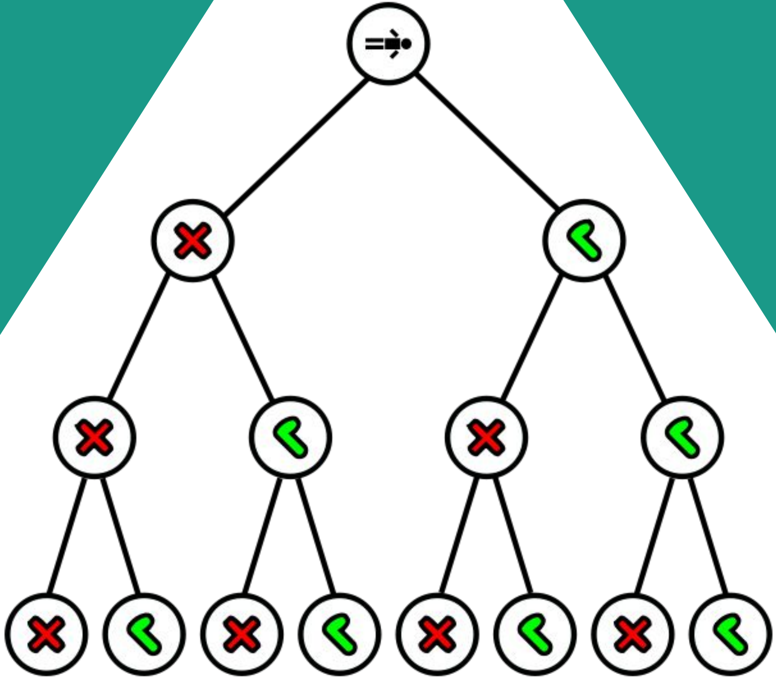
- ➔ `==` checks if two objects have the same content
- ➔ **is** checks if two variables point to the same object
  - ◆ **Booleans**, **Numbers**, **Strings**, and **None** all share the same object in memory
  - ◆ Use **==** to compare **numbers** or **string** literals

Logical and Membership Operators

Operator	Description	Example of True
`and`	Returns True if both expressions are true	True `and` True
`or`	Returns True if at least one expression is true	True `or` False
`not`	Reverses a booleans	`not` (False)

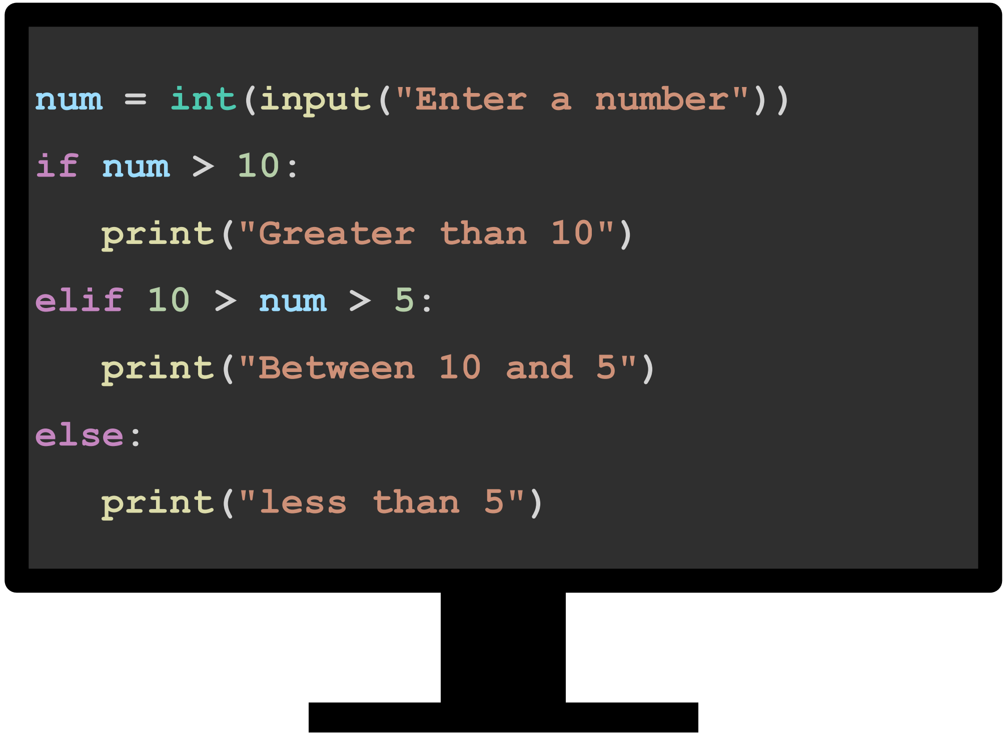
Operator	Description	Example of True
`in`	Returns True if a sequence with the specified value is present in the object	3 `in` [1,2,3]
`not in`	Returns True if a sequence with the specified value is not present	5 `not in` [1,2,3]

Conditionals



if/elif/else

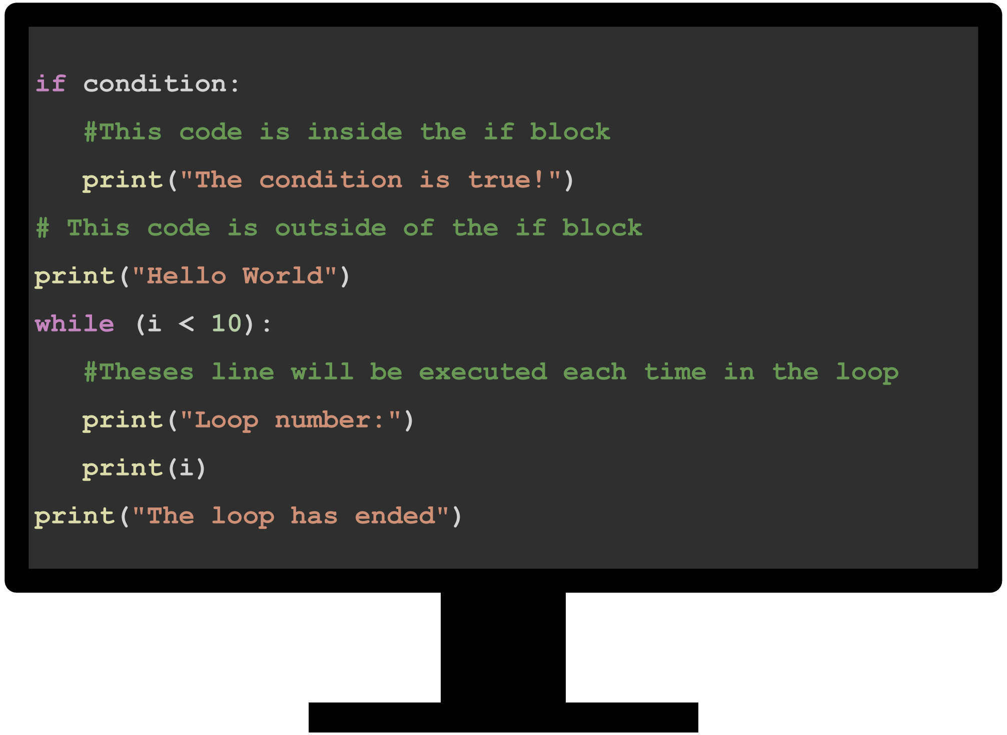
- Python's Else If syntax is called **elif**
- Any condition that resolved to True will cause all statements within the if block to execute.
  - Remember that consistent indentation counts as a block



```
num = int(input("Enter a number"))
if num > 10:
    print("Greater than 10")
elif 10 > num > 5:
    print("Between 10 and 5")
else:
    print("less than 5")
```

Indentation

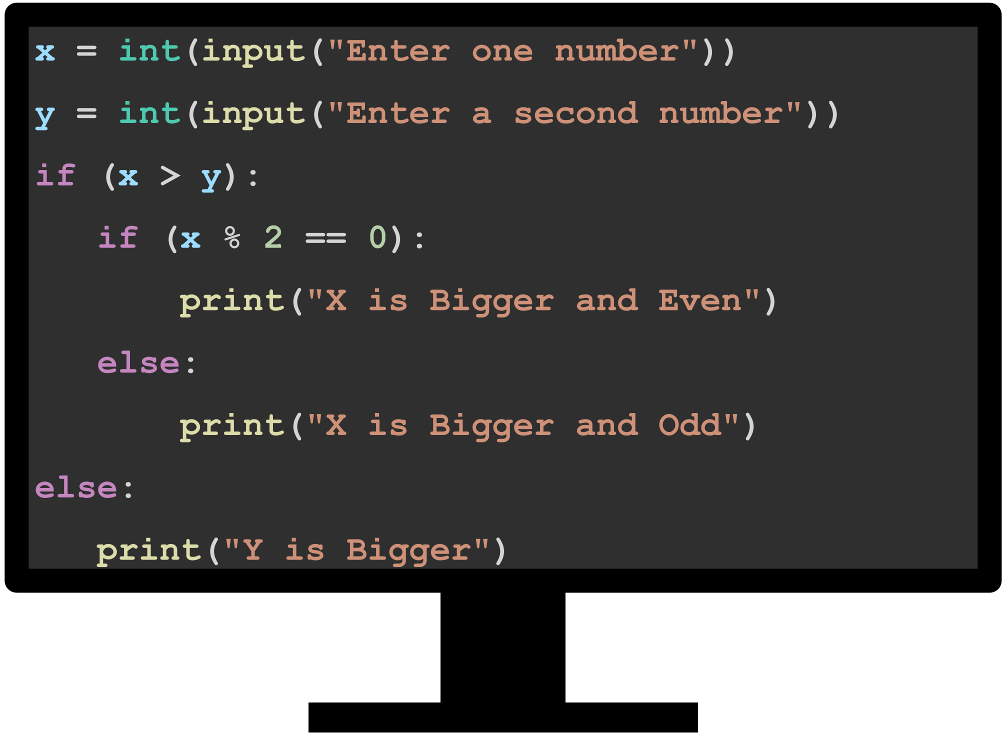
- ➔ Python uses **indentation** and whitespace to define **code blocks**
  - ◆ Many languages use the curly brackets for this { }
- ➔ A colon : is used to instantiate a **block**
- ➔ The number of spaces per **indent** is up to the developer, but it must be consistent



```
if condition:
    #This code is inside the if block
    print("The condition is true!")
# This code is outside of the if block
print("Hello World")
while (i < 10):
    #Theses line will be executed each time in the loop
    print("Loop number:")
    print(i)
print("The loop has ended")
```

Nesting ifs

- ➔ Each indented block is considered its own **block**
- ➔ If statements can be nested within other if statements to chain conditionals



```
x = int(input("Enter one number"))
y = int(input("Enter a second number"))
if (x > y):
    if (x % 2 == 0):
        print("X is Bigger and Even")
    else:
        print("X is Bigger and Odd")
else:
    print("Y is Bigger")
```

Single Line If

- ➔ A single line conditional does not need an indent

```python
if x > y: print("x is greater than y")
elif x < y: print("y is greater than x")
else: print("x is equal to y")
```

- ➔ **Ternary Operators** set a value based on a condition
    - ◆ Often known as **Conditional Expressions**

```python
message = "Greater than 10" if  (num > 10) else "Less than 10"
```

Pass Statement

➔ Python code blocks cannot be empty

```
if i is 10:

print("Empty condition") # Error!
```

➔ The **pass** statement will allow an empty block to be skipped

```
if i is 10:
  pass
print("Empty condition") # Success!
```

Student Exercise

- ➔ Rock - Paper - Scissors
  - ◆ Write a program that accepts the user's input, and make sure that input is either **r**, **p**, or **s**
  - ◆ Ask for a second user's input, and make sure that input is also only **r**, **p**, or **s**
  - ◆ Complete the program to output the results of a rock, paper, scissors game
  - ◆ The game only needs to run once