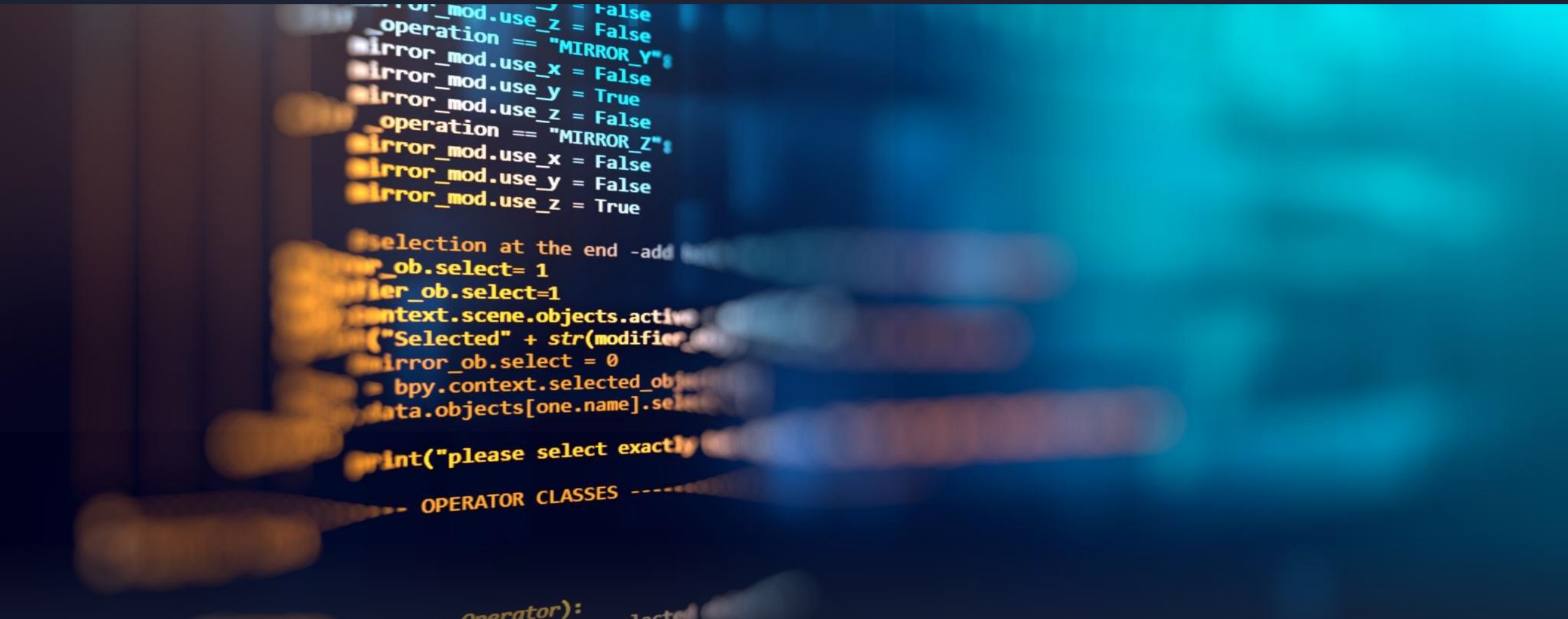
Introduction to Python Programming

BEAUTY AND JOY OF COMPUTING

- BJC GROUP



Why Python?

- What is Python? Is it a robot snake?
- Python is an interpreted, high-level programming language.
- It was developed by Guiddo van Rossum in 1991, who named it Python after inspired his favorite British Comedy Group " Monty Python's Flying Circus".
- Python emphasizes on simplicity and rapid development by offering a huge library of **modules**, reusable blocks of code written by someone else, so you don't reinvent the wheel.

Python's key features

- Easy to learn: The syntax is human friendly as it resembles English.
- Interpreted language: Most programming languages need to be compiled to machine code before executing the programming while python's interpreter executes code line by line.
- Dynamic Typing: No need to explicitly define types.
- Extensive libraries: Offers vast built-in-libraries for applications such as machine learning, web development and data science.

Setting up Python

- Now to fun part!
- Install python from https://www.python.org/
- Install an IDE (Integrated Development environment) of your choice; we recommend Visual Studio Code.
- Alternatively, an online Python IDE is available at https://www.online-python.com/ (Doesn't require installation, however it has limited capabilities)

**IDEs offer a usable environment and tools for assisiting with writing code. **

First Python Code

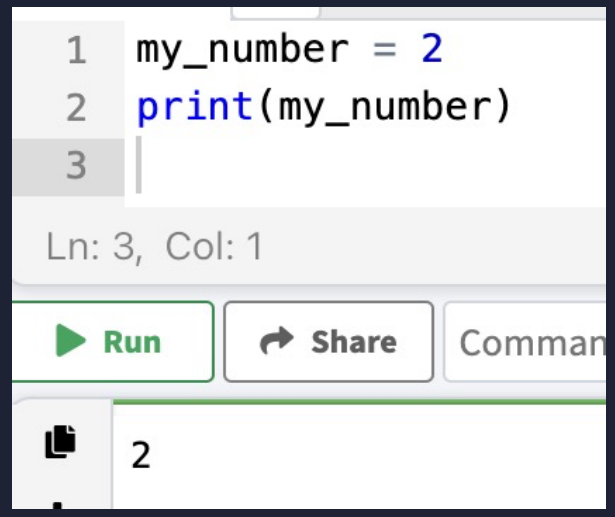
- Write this code your IDE:

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

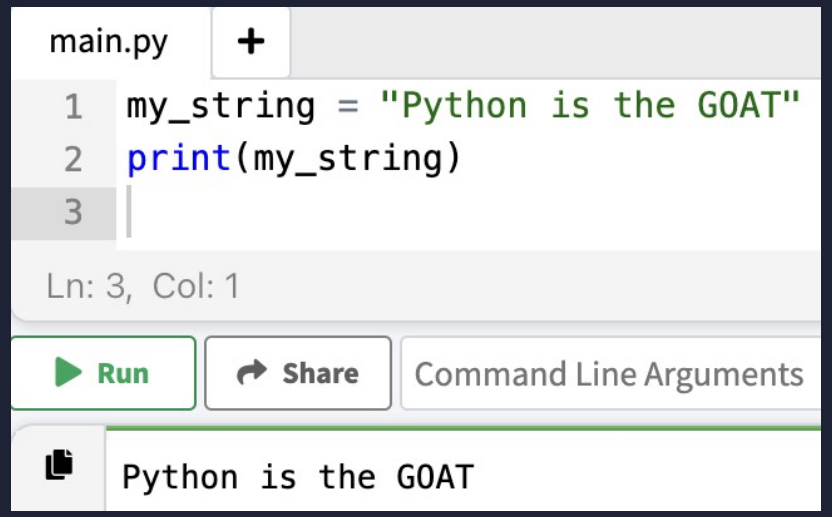
- On your IDE's terminal, you should see "Hello World" displayed.
- As you might have guessed the "print" is a built-in function that displays the output of what is wrapped within the opening & closing parenthesis.
- You also notice that "Hello, world" is wrapped with double quotation marks. This is called a string literal and it's one of the data types covered in this course.

Variables & data types in python

- Variables in programming are abstract containers for storing data values.
- Variables in python are named using camel case convention ("_" after every word)
- Data types are attributes that help the computer determine the type of the data and what functionalities can be performed on it.
- In python "=" is used to assign a value (right side) to a variable (left side of equal sign).
- Data types in python:
  - Numeric: `int`, `float`, `complex`
  - Text: `str`
  - Sequence: `list`, `tuple`, `range`
  - Mapping: `dict`
  - Set: `set`, `frozenset`
  - Boolean: `bool`



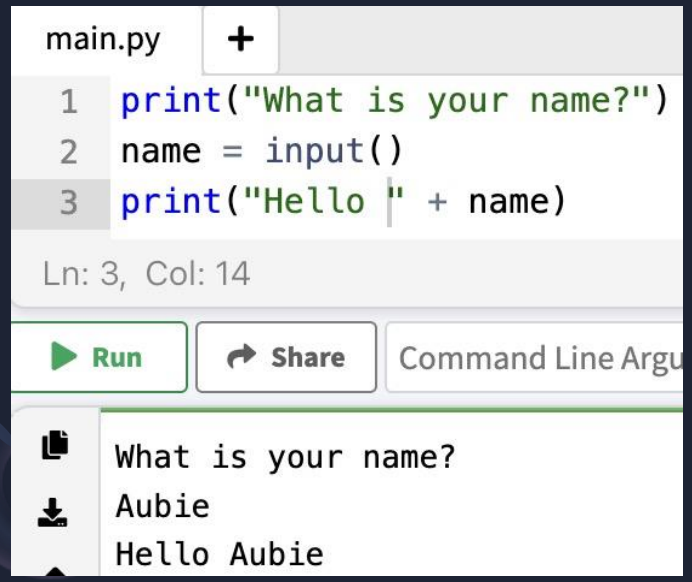
Using an int variable



Using a string variable

Input and output

- What if we want the program to not just hello world but to greet us?
- Python's attribute "input" prompts the user for some input in the terminal.



Operators and Expressions

- Operators in Python:
  - + (Addition): x + y
  - - (Subtraction): x - y
  - * (Multiplication): x * y
  - / (Division): x / y
  - // (Floor Division): x // y
  - % (Modulus): x % y
  - ** (Exponentiation): x ** y

- Expressions in Python
  - They are a combination of values, variables and operators.
  - It's usually the right side of the "=" sign assignment.

```
1 a = 1
2 b = 1
3 answer = a + b
4 print("A very complex math: 1 + 1 = " + str(answer))
```

Ln: 4, Col: 53

Run | Share | Command Line Arguments

```
A very complex math: 1 + 1 = 2
```

Str() attribute is used to convert an integer into a string type, that way we can append the "int answer" to the string literal

NOTE: In programming, the "=" sign by itself is used to assign values to a variable.
If attempting to perform an equality comparison, "==" is used instead.

Control Flow (If-else statements)

- If statements: A control flow statement to execute code based on conditions

main.py

```
1  animal = "cat"
2
3  if animal == "cat":
4      print("meouw")
5  elif animal == "dog":
6      print("woof woof")
7  else:
8      print("animal sounds")
```

Ln: 8, Col: 27

Run | Share | Command Line Argume

```
meouw
```

Data Structures (Arrays/Lists)

- Arrays: A sequence of characters (any type) stored in an indexable, contigious memory block.
- Strings, for example "hello, world", are immutable arrays (Once created, can index elements but can't change them.
- Lists: In python they an array implementation that's dynamic and comes with a collection of attributes.
- 0-Indexed using brackets "[]". To extract the first element of my_list, write this code: my_list[0]

Data Structures (List attributes)

- Len(): returns the number of items in the list

main.py

```
weather = ["sunny", "rainy", 100]
weather_count = len(weather)
print(weather_count)
```

Ln: 3, Col: 21

Run | Share | Command Line Arguments

```
3
```

- Sort(): sorts the list **in-place** (modifies the original list).
- Check out more on lists: https://www.w3schools.com/python/python_lists.asp

Data Structures (Tuples and Sets)

- Tuples are immutable, ordered collection of elements.
- Features:
  - Cannot be modified (no add, remove, or update).
  - Allows duplicate elements.
  - Indexed and can be sliced.

```
my_tuple = (3,1,2,3)
```

- Helpful when working with groups of data like coordinates (x, y).

Data Structures (Sets)

- Sets are unordered, mutable collection of unique elements.
- Features:
  - No duplicate elements.
  - Unordered, so indexing is not possible.
  - Supports set operations like union and intersection.

```
1  my_set = {"apple", "banana", "apple", "cherry"}
2  print(my_set)
```

Ln: 2, Col: 14

Run | Share | Command Line Arguments

```
{'apple', 'banana', 'cherry'}
```

- Carly braces "{}" are used to notate sets in python.

Data Structures (Dictionaries)

- Dictionaries are mutable, unordered collection of key-value pairs.
- Features:
  - Keys must be unique and immutable.
  - Values can be of any data type.
  - Allows fast lookups by key.

```
my_dictionary = {"name": "John", "age": 20}
print(my_dictionary["name"])
```

Ln: 2, Col: 29

Run | Share | Command Line Arguments

```
John
```

- Notated with "{}" like sets. The difference is that dictionaries use key-value pairs.
- Example use case: used to represent real world objects with properties.

Functions

What is a Function?

- A reusable block of code designed to perform a specific task.
- Helps to organize code and improve readability.

Features:

- **Encapsulation**: Wrap code into a callable unit.
- **Modularity**: Divide a program into smaller, manageable parts.
- **Reusability**: Call a function multiple times.
- **Avoid Redundancy**: Reduce code duplication.

Functions

- **Built-in Functions**: Predefined in Python.

  Example: len(), print(), sum().

- **User-defined Functions**: Created by the user.

- **Anonymous Functions**: Using lambda.

```python
square = lambda x: x**2
two_sum = lambda x,y: x+y
print(square(2)) # Will pass in the parameter as the x value
print(two_sum(2,3))
```

Ln: 5, Col: 1

Run | Share | Command Line Arguments

```
4
5
```

Anonymous function example

```python
def greet(name):
    print("Hello, " + str(name))
greet("Lebron James")
greet("Stephen Curry")
```

Ln: 5, Col: 1

Run | Share | Command Line Arguments

```
Hello, Lebron James
Hello, Stephen Curry
```

User-defined function example

Python Modules

What is a Module?

- A file containing Python code (functions, classes, variables).
- Helps in organizing and reusing code across programs.

Features

- **Code Reusability**: Import and reuse functionality.
- **Namespace Management**: Avoid name conflicts.
- **Built-in and Custom Modules**: Predefined (e.g., math, random) or user-created.

Importing Python Modules

- Importing entire module

```python
import math
print(math.sqrt(16))
```

- Importing specific functions

```python
from math import sqrt
print(sqrt(16))
```

- Importing with an alias

```python
from math as m
print(m.sqrt(16))
```

- To create your own module:
  - Save functions in a file (e.g., my_module.py)

```python
import my_module
my_module.my_function()
```

Make sure to save to a file in the same folder containing the import

Wrapping up

There you have it. Python is not a robot snake but a powerful and versatile programming language that offers countless tools to help with your programming endeavours.

A few tips:

- Start with honing down the basics.
- Work on projects as you learn. Learning is reinforced by doing and don't be afraid to mess up as that's part of the learning.
- Errors or bugs are inevitable. Writing down comments and using descriptive names for your variables can help with the debuggging process.

Resources to guide your learning

Check out this resources for additional learning:

W3schools website offers examples and an interactive GUI to test out examples

- https://www.w3schools.com/python/python_intro.asp