BASIC PROGRAMMING

PREPARED BY: LUIS MEING

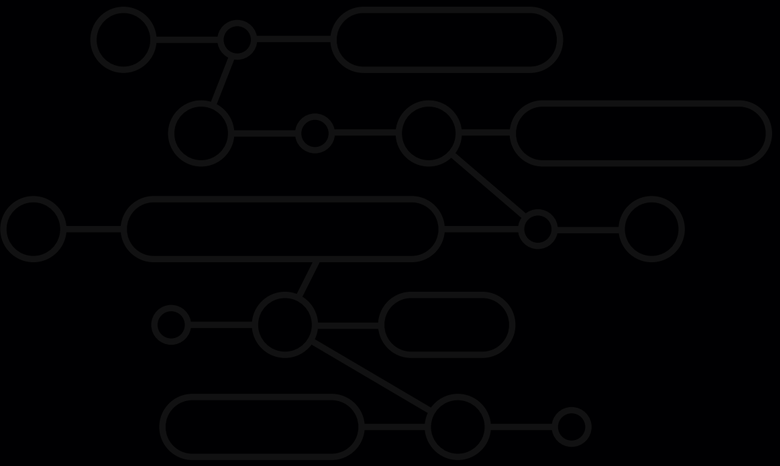
OBJECTIVES:

01

Introduce Programming

1.1: Be acquainted with Python

1.2: Know fundamentals of a program and programming language



What is Python?

It is an **interpreted**, **object-oriented**, **high-level programming language** with **dynamic semantics**.

"interpretted"

"It is where the source code is not directly translated by the target machine"

The language you understand is not the languge the computer understands and vice versa

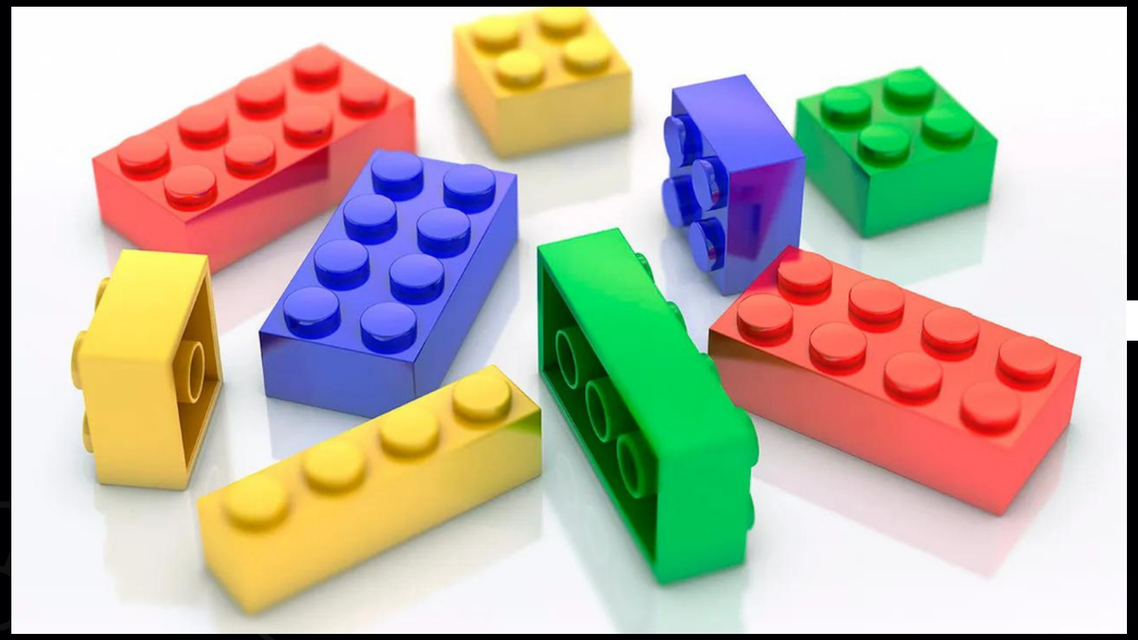
"object-oriented"

"It is a computer programming model that organizes software design around data, or objects, rather than functions and logic."

Blocks and boxes can be seen everywhere

1.1 Be acquainted with Python

Commands, variables, etc

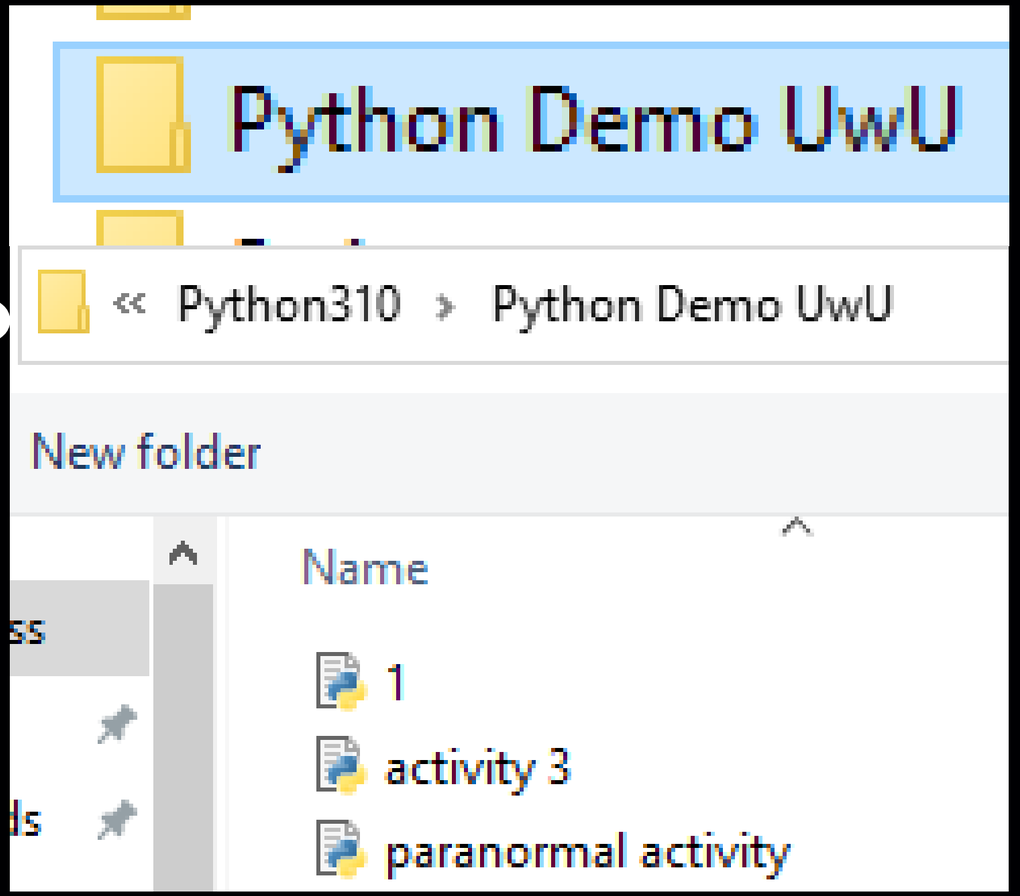


programs

1.1 Be acquainted with Python

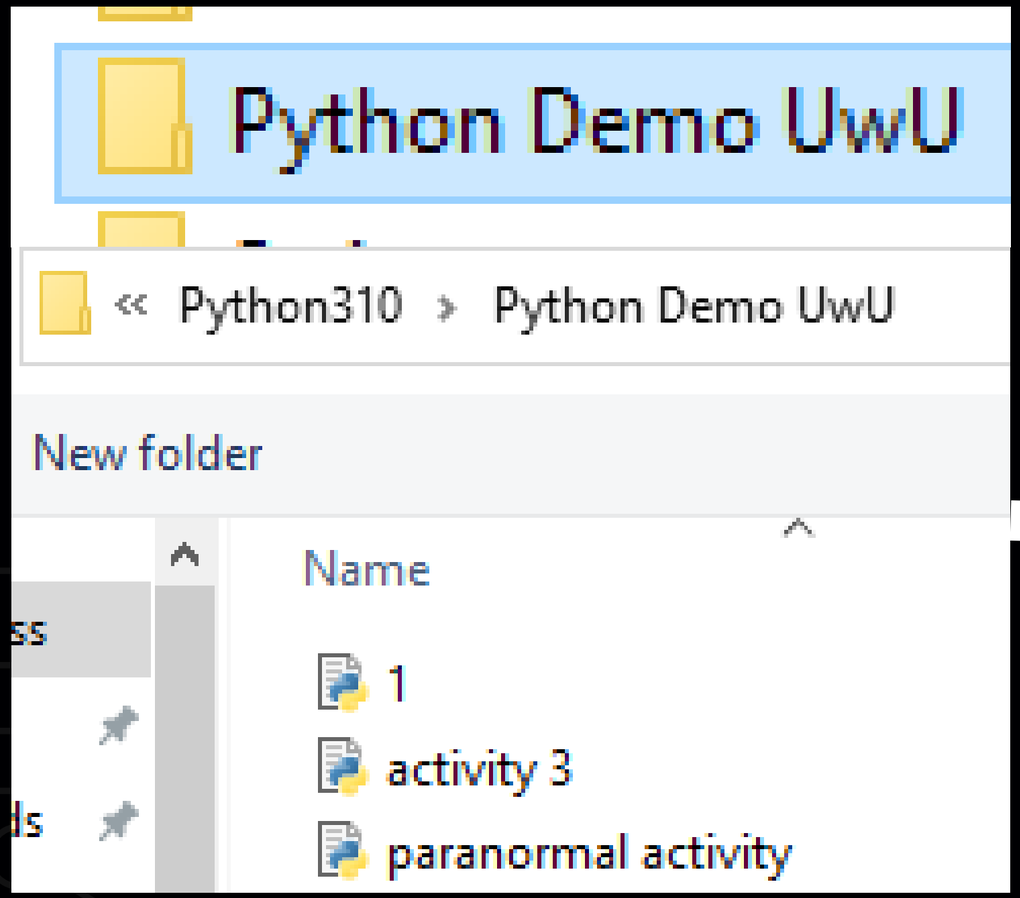
programs

Project

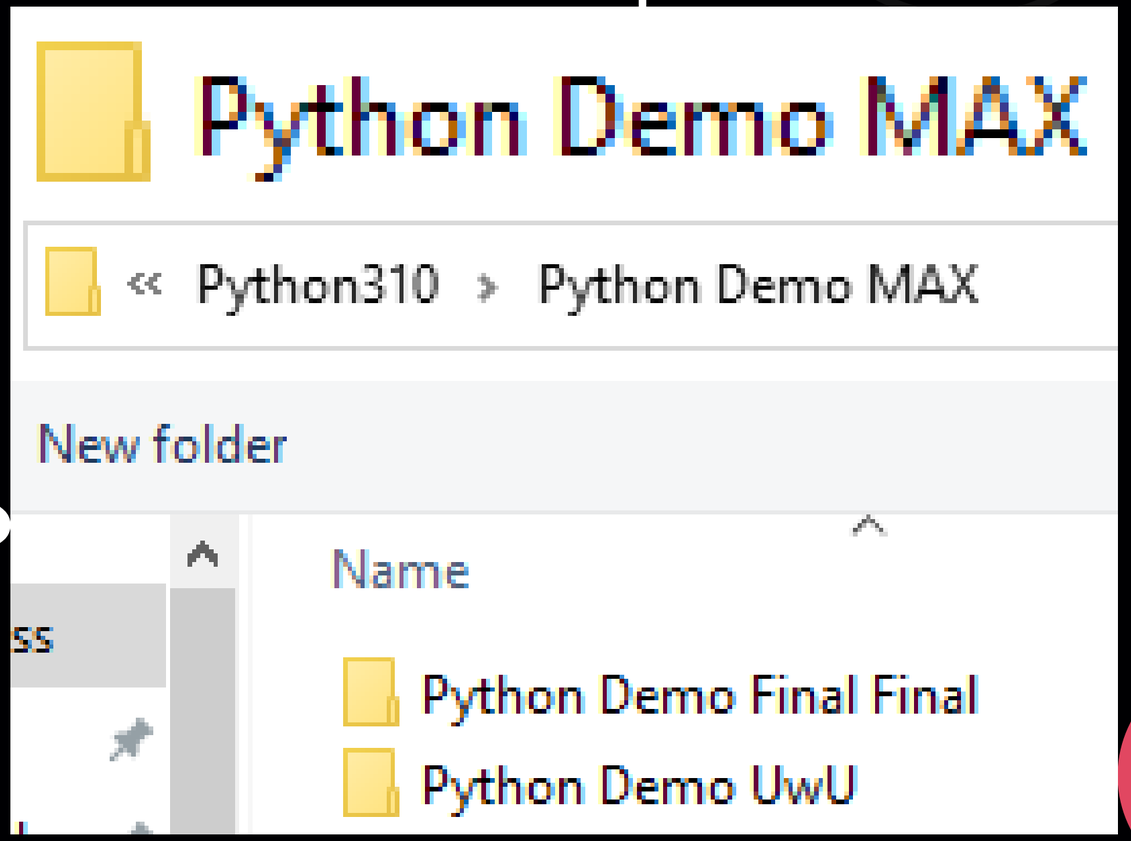


1.1 Be acquainted with Python

Project



Bigger Project/Program Group



"high-level programming language"

"It is any programming language that is user-friendly for programming and is generally independent of the computer's hardware architecture"

Can communicate with all types of computers

"dynamic semantics"

"It is an approach to defining where pieces of text are viewed as instructions to update an existing context, the result of which is an updated context."

Values of variables can change

Print Command

syntax:

print (<add something here>)

```
>>> print ('hello world')
    hello world
```

Variables

- variables are names that can be assigned a value and then used to refer to that value throughout your code

It is a labelled box that can store value

Variables

Declaring/Summoning/Creating/Using/Syntax:

<variable name> = <value>

```
student_id = 123456
```

Variables

Can it values be overwritten/replaced?

<variable name> = <value>

Variables

Can it values be overwritten/replaced?

<variable name> = <value>

```
>>> student_id= 'meing123'
```

How do I check if it worked?

1.2.2.1 Know fundamentals of a program and programming language

```
>>> student_id = 123456
>>> print (student_id)
123456
>>> student_id= 'meing123'
>>> print (student_id)
meing123
```

Rules of Variable Naming

1. Can be as long or as short as you like
2. May contain uppercase and lowercase letters (A – Z, a – z), digits (0 – 9), and underscores (__)
3. Start with a letter or the underscore character
4. Variable names are case-sensitive (must be named or referred to in the identical fashion)
5. Use very descriptive names

Variable Naming Conventions

- Camel Case naming convention
  - myAge
  - favColorInTheColorWheel
- Pascal Case naming convention
  - MyAge
  - FavColorInTheColorWheel
- Snake Case naming convention
  - my_age
  - fav_color_in_the_color_wheel

Variable Data Types

- Are the things/values that a variable can have
- These are:
  - *Integer*
  - *Floating Point*
  - *String*
  - *Boolean*
  - *Null*

Integers

- These are zero, positive, negative, whole numbers.
- There is no explicitly defined limit in the value of the integer Python.

```
student_id = 123456
```

Float

- This represents a floating-point number.
- They are represented with a decimal point.
  - x = 9.999999999999
- You can also make use of negative values for the floating-point number.
  - x = -73.435

```
student_id = 1.2345
```

String

- Strings are a sequence of bytes representing Unicode characters.
- There are several ways to create a string.
- They differ based on the delimiters and whether a string is single or multiline.

```
student_id= 'meing123'
```

String Delimitter

- Can use double quotes ("<any>") or single ('<any>') quotes
  - Is useful for creating sentences containing single quotes
  - To get around this, the "\" can be used.
- For longer strings, you can use the triple quotes ("'<any>'")

Know fundamentals of a program and programming language

String

```
>>> student_id = 1.2345
>>> print ("Louie's Report")
    Louie's Report
>>> print ('Louie\'s Report')
    Louie's Report
>>> print ('Louie's Report')
...
    SyntaxError: unterminated string
    literal (detected at line 1)
```

String

- Concatenation ("+")
- Can be used to stitch together strings and values

```
>>> my_name="Louie"
>>> print ("Hello " + my_name)
Hello Louie
```

Boolean

- Boolean data types determine the truth value of expressions.
- They can either be True or False.
  - booleanVal = True
  - print(booleanVal) # Outputs True

Know fundamentals of a program and programming language

Boolean

```
>>> is_a_student = true
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <
module>
    is_a_student = true
NameError: name 'true' is not defi
ned. Did you mean: 'True'?
>>> is_a_student = True
```

Null

- This type is used to define a null variable or object.
- It makes use of the “None” keyword.
- We can assign “None” to any variable.
- It can also be used in Expressions.
  - emptyVal = None
  - print(emptyVal) # Outputs None

1.2.2.
4.5 Know fundamentals of a program and programming language

Null

```
>>> is_a_student = True
>>> has_a_section = none
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <
module>
    has_a_section = none
NameError: name 'none' is not defi
ned. Did you mean: 'None'?
>>> has_a_section = None
>>>
```

Comments

Syntax:

#<anything>

```
>>> #this line is a comment and       will
    not be used
>>> #student_id=123meing
>>> print (student_id) #print ('hello')
    meing123
```

? Exercise

Output is?

```
>>> print ('Hello!')
```

? Exercise

Output is?

```
>>> print ('Hello!')
```

```
Hello!
```

? Exercise

Output is?

```
print #hello ('world')
```

? Exercise

Output is?

```
print #hello ('world')
```

```
>>> print #hello ('world')
<built-in function print>
```

"None"

? Exercise

Output is?

```
your_name = 'Jaughn'
```

? Exercise

Output is?

```
your_name = 'Jaughn'
```

None, only a variable was created

? Exercise

Output is?

```
your_name = Jaughn
print (your_name)
```

? Exercise

Output is?

```
your_name = Jaughn
print (your_name)
```

```
your_name = Jaughn
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    your_name = Jaughn
NameError: name 'Jaughn' is not defined
```

? Exercise

Output is?

```
your_name = "jaughn"
print ('your_name')
```

? Exercise

Output is?

```
your_name = "jaughn"
print ('your_name')
```

```
>>> your_name = "jaughn"
>>> print ('your_name')
your_name
```

? Exercise

Output is?

```
your_name = "jaughn"
print ("I'm pleased to meet you" + your name)
```

? Exercise

Output is?

```
your_name = "jaughn"
print ("I'm pleased to meet you" + your name)
```

```
>>> print ("I'm pleased to meet you" + your name)
    SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

? Exercise

Output is?

```
print ("I'm pleased to meet you" + your_name)
```

? Exercise

Output is?

```
print ("I'm pleased to meet you" + your_name)
```

```
>>> print ("I'm pleased to meet you" + your_name)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print ("I'm pleased to meet you" + your_name)
NameError: name 'your_name' is not defined
```

? Exercise

Output is?

```
one_dollar = 50
2_dollars = 100
```

? Exercise

Output is?

```
one_dollar = 50
2_dollars = 100
```

```
>>> one_dollar = 50
>>> 2_dollars = 100
    SyntaxError: invalid decimal literal
```

? Exercise

Output is?

```
your_name = "Jaughn"
print ("I'm pleased to meet you" + your_name)
```

? Exercise

Output is?

```
your_name = "Jaughn"
print ("I'm pleased to meet you" + your_name)
```

```
>>> your_name = "Jaughn"
>>> print ("I'm pleased to meet you" + your_name)
I'm pleased to meet youJaughn
```

? Exercise

Output is?

```
your_name = "Jaughn"
print ("I'm pleased to meet you " + your_name)
```

? Exercise

Output is?

```
your_name = "Jaughn"
print ("I'm pleased to meet you " + your_name)
```

```
>>> your_name = "Jaughn"
>>> print ("I'm pleased to meet you " + your_name)
I'm pleased to meet you Jaughn
```