What is Python?

Python is a popular programming language. It was created by **Guido van Rossum**, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different **platforms** (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax **similar to the English** language.
- Python has syntax that allows developers to write programs with **fewer lines** than some other programming languages.
- Python runs on an **interpreter** system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a **procedura**l way, an **object-oriented** way or a **functional** way.Good to know
- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment **(IDE)**, such as **Thonny**, **Pycharm**, **Netbeans** or **Eclipse** which are particularly useful when managing larger collections of Python files.Python Syntax compared to other programming languages
- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on **indentation**, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Comments in Python:

-python commets started with a #

```
#This is a comment
print("Hello, World!")
```

Creating Variables
Python has no command for declaring a variable.
A variable is created the moment you first assign a value to it.

```
x = 5
y = "John"
print(x)
print(y)

a=10
b=20
c=a+b
print (c)
```

Casting

If you want to specify the data type of a variable, this can be done with casting.

Example

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
a=int(input("Enter the value : "))
```

Get the Type
You can get the data type of a variable with the **type()** function.

```
type(x)  # it will return str
```

Case Sensitivity: the variables are case sensitive

```
a=10
A=20
print (a)
print(A)
```

Variable Names
A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
Rules for Python variables:

- A variable name must start with a letter(a) or the underscore(_a) character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Multi Words Variable Names
Variable names with more than one word can be difficult to read.
There are several techniques you can use to make them more readable:
-Camel Case
Each word, except the first, starts with a capital letter:
```
myVariableName = "John"
```
-Pascal Case
Each word starts with a capital letter:
```
MyVariableName = "John"
```
-Snake Case
Each word is separated by an underscore character:
```
my_variable_name = "John"
```

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

One value to Multiple Variable:

```
x=y=z="Orange"
print(x)
print(y)
print(z)
```

Unpack a Collection
If you have a collection of values in a list, tuple etc. Python allows you extract the values into variables. This is called *unpacking*.

Example
Unpack a list:

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

There are techniques to print output of a variable:

Ex1:

```
a=10
print(a)
print("The value of a is: ", a)
```

Ex2:

```
a=10
b=20
print(a+b)
c=a+b
print(c)
print("the sum of ", a, "and ", b, "is: ", c )
```

Ex3:

```
a=10
b="A"
c=a+b
print(c)
```

this will show some error. But if we want to add two or more string-

```
a="A"
b="B"
print(a+b)
```

this is valid.

Scope of a variable in python:

The value of a function creted inside any block has the scope only to that block. otherwisethe global value is appeared.

```
a=10
def func():
    print(a)

func()
```

it will print the given value of a

lets have another example-

```
a=10
def func():
    a=100
    print("Inside value of a is: "a)
print("Outside value of a is: ",a)
func()
------------------------------
```

```
Outside value of a is:  10
Inside value of a is:  100
```

***global keyword:** if we want to create a global variable from inside of a block

```
a=10
b=20
def func():
        a=100
        global b
        b=200
        print("Inside value of a  is: ", a, "and b is: ",b)
func()
print("Outside value of a is: ",a, "and b is: ",b)
```

```
Inside value of a  is:  100 and b is:  200
Outside value of a is:  10 and b is:  200
```

Python has the following data types
built-in by default, in these categories:
Text Type:str
Numeric Types :int, float, complex
Sequence Types: list, tuple, range
Mapping Type: dict
Set Types :set, frozenset
Boolean Type :bool
Binary Types: bytes, bytearray, memoryview

-we can check the type of any data by **type()** function

```
a=10
print(type(a))
```

The data types are automatically be assigned when we put some value on it.

Setting the Specific Data Type

```
a=int(input("Enter some value: "))
```

If you want to specify the data type, you can use the following constructor functions:

Example

```
x = str("Hello World")
 »x = int(20)
 »x = float(20.5)
 »x = complex(1j)
  »x = list(("apple", "banana", "cherry"))
   »x = tuple(("apple", "banana", "cherry"))
    »x = range(6)
    »x = dict(name="John", age=36)
    »x = set(("apple", "banana", "cherry"))
    »x = frozenset(("apple", "banana", "cherry"))
    »x = bool(5)
    »x = bytes(5)
    »x = bytearray(5)
    »x = memoryview(bytes(5))
```

DATA TYPES AND OPERATIONS

- Numbers
-string
-List
-Tuple
-Set
-Dictionary

NUMBERS:

basic four numerical data type

-int (the length can be any depending upon the available memory

-long

-float

-compex

Mathematical Functions:

abs(x)- absolute value

Task: Take input of any number and find its absolute value

sqrt()

ceil(x)--- the smallent integer after x

floor(x)-largest integer before x

pow(x,y)--tell me the meaningV

exp(x)

fabs(x)---absolute value

log(x)

log10(x)--base 10logarithm of x

max(x1, x2, ...)

min(x1, x2, ...)-

round(x,[n])---x is a decimal numbe, after decimal n digits will be there

modf(x)- if separates decimal and integer part of a number and represent them in decimal number in a tuple

modf(10.01)- it will return(0.01, 10.0)

Some trigonometrical Functions()

-sin(x)

-cos(x)

-tan(x)

-asin(x)

-acos(x)

-atan(x)

-atan2(y,x)----it returns atan(y/x)in radians

-hypot(x,y)- Euclidean form, sqrt(x*x+y*y)

-degrees(x)---radian to degree conversion

radians(x)--converts degree to radian

Task: Take external input and test all those trigonometrical functions

some randon number function:

-choice(sequence)- creates some random value in a sequence (like list, tuple, string)

shuffle(list)- randomize the elements of a list

random()- random float between 0-1

randrange([start],[stop],step)- random numbers in a range between 'start' and 'stop' and the gap decided by 'step'

seed([x]) - starting value of generating random number. this functi is generally called before calling other random module functions.

uniform(x,y)- generates a random floating point number in between x to y

Task: Take external input and test all those random number functions

Escape character:

\a- Bell or alert
\b- backspace
\f- formfeed
\n-
\r- carriage return
\s- space
\t- tab
\v - vertical tab

Format (control string) symbols:

```
a=10
b=20
c=a+b
print("The sum of %d and %d is: %d" %a, %b, %c)
```

NOTE:there is no comma after double quote

%c- character
%s- string
%i- signed decimal integer
%d- same
%u- unsigned integer
%o- for octal integer
%-x - small x for hexadecimal integer (lowercase)
%-X - capital X for hexadecimal integer (uppercase)
%e- exponential notation with lowercase letter e
%E- exponential notation with upperrcase letter E
%f- floating point number

Type Casting:

Specify a Variable Type
There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using **constructor functions**:

- **int()** - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- **float()** - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- **str()** - constructs a string from a wide variety of data types, including strings, integer literals and float literals

String operations

1. Lenth of a string- len(string)

Ex:-

```
s="welcome to Python"
print("Lenth of ", s, "is: ", len(s))
```

2. lower(): converts every letter of thestring to lower case

Ex:

```
   s="Wecome to python"
   print(s.lower())
```

3. upper(): converts every letter of thestring to upper case

Ex:

```
   s="Wecome to python"
   print(s.upper())
```

4. swapcase(): swap letters to upper to lower and vice versa

Ex:

```
   s="Wecome to python"
   print(s.swapcase())
```

5. capitalize(): the first letter became capital

Ex:

```
   s="Wecome to python"
   print(s.capittalize())
```

6. title(): all words of the string gets capitalized

Ex:

```
   s="Wecome to python"
   print(s.capittalize())
```

7. Istrip(): all the character from beggining are removed

Ex:

```
   s="----- Wecome to Python"
   print(s.lstrip("-"))
```

8. rstrip(): all the characters are removed from end

Ex:

```
   s=" Wecome to Python-----"
   print(s.rstrip("-"))
```

9. strip(): all the characters are removed from start and end

Ex:

```
   s="---- Wecome to Python-----"
   print(s.strip("-"))
```

10. min(string) and max(string): minimum and maximum alphabetical character fron the string

Ex:

```
   s="Wecome to python"
   print("Minimum= ", min(s))----------- this will give blank space as output
   print("Maximum=:", max(s))
```

11. count(idetifier, beg=0,end=len(string)): count of occurance of the identifier (alphabet) in a given string

Ex:

```
s="Wecome to python"
print(s.count('o', 0, 5))
print(s.count('o', 0, 10))
print(s.count('o', 5, 10))
print(s.count('o',0, len(s)))
```

12. index(idetifier, beg=0,end=len(string)) and rindex(): it will check the identifier is there or not, if not found it ll give some error.rindex() does the same but from backward

Ex:

```
s="Wecome to python"
print(s.count('the', 0, len(s)))
print(s.rindex('to', 0, len(s)))
```

13. find(idetifier, beg=0,end=len(string)) and rfind(idetifier, beg=0,end=len(string)): index of the identifier if it is found and gives -1 if not found. rfind() does the same but from backward

Ex:

```
s="Wecome to python"
print(s.find('to', 0, len(s)))
print(s.rfind('to', 0, len(s)))
```

14. join(sequence): joins the strings as they comes in sequence.

Ex:

```
s='-'
sequence=("Hello", "world", "!")
print(s.join(sequence))
```

output will be: Hello- world-!

Basic data type structures and if possible control structure also