This Document Contains {Day1} BASICS OF PYTHON by Rupam Gupta.

TOPICS COVERED= INTRO, SYNTAX, QUOTATIONS, EXCEPTIONAL CASE, BASIC OPERATIONS, VARIABLES, DATA TYPES, CONCATENATION & FORMATING.

WHY PYTHON?

1) beginner friendly.

2) compatible.

3) vast libraries & open source framework & tool.

4) more readability.

5) interactive mode.

Python is case sensitive.

Python was developed by **Guido van Rossum** in the early **1990s** and its **latest version is 3.11.0**, we can simply call it Python3.

Python 3.0 was released in **2008**, and is interpreted language i.e it's not compiled and the interpreter will check the code line by line

Inorder to print anything in PYTHON, use the print statement & write **print()**

usage of paranthesis () is must, else it will return error

In [14]:
```
print("hello world") #to print any string we need to enclose them with double inverted commas ie. (" ")
```

```
hello world
```

In [15]:
```
print("welcome to the day 1 of Basics Python")
```

```
welcome to the day 1 of Basics Python
```

To add new cell, use shortcut :- **shift + enter**

To run the code, use shortcut :- **ctrl + enter**

Printing the multi line code - use triple quotation (""" """)

In [16]:
```
print("""hello Beginners
Python is super easy to learn""")
```

```
hello Beginners
Python is super easy to learn
```

using triple quotation help to print multiline statements easily or else we have to execute each lines one after other as following

In [25]:
```
print("hello beginners")
print("Python is super easy to learn")
```

```
hello beginners
Python is super easy to learn
```

more exceptional cases in print statements

Sometimes suppose we want to highlight any internal element from the **"string"**, then we have to enclose that **'internal element'** to with different quotation.

lets see that with an example

In [26]:
```
print("Hi I am a "GEEK" ") #this will return error, since external & internal quotation are same.
```

```
  File "C:\Users\abc\AppData\Local\Temp/ipykernel_15804/792815677.py", line 1
    print("Hi my name is "Sunny" ") #this will return error, since external & internal quotation are same.
                          ^
SyntaxError: invalid syntax
```

In [27]:
```
#let modify the above test case.

print("Hi I am a 'GEEK' ")
```

```
Hi I am a 'GEEK'
```

In [30]:
```
print("""Hi I am a "GEEK" """)
```

```
Hi I am a "GEEK"
```

In [31]:
```
print("""Hi I am a "GEEK"! """)
```

```
Hi I am a "GEEK"!
```

Lets learn some basic operations in Python

In [32]:
```
#addition

241+1234
```

Out[32]: 1475

In [33]:
```
#substraction

4509-1314
```

Out[33]: 3195

In [34]:
```
#multiplication

35*23
```

Out[34]: 805

In [36]:
```
#division

24/50
```

Out[36]: 0.48

In [37]:
```
#power of any number

35**2 #square
```

Out[37]: 1225

In [42]:
```
45**3 #cube
```

Out[42]: 91125

In [46]:
```
#modulus, returns the reminder

98%4 #remainder is 2
```

Out[46]: 2

In [48]:
```
5297%6
```

Out[48]: 5

What is a Variable in Python?

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing.

Variables are containers for storing data values.

Python Variable Types

Every value in Python has a datatype. Different data types in Python are **Numbers, List, Tuple, Strings, Dictionary, etc.** Variables in Python can be declared by any name or even alphabets like a, aa, abc, etc.

Rules for Python variables:

A variable name must start with a **letter** or the **underscore** 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)**

```
#Legal variable names:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

#Illegal variable names:
2myvar = "John"
my-var = "John"
my var = "John"
```

lets try out some examples of the variables

In [49]:
```
my_var= 25
```

In [50]:
```
print(my_var)
```

```
25
```

In [51]:
```
my_var2= 35
```

In [52]:
```
print(my_var2)
```

```
35
```

In [55]:
```
add_var=(my_var+my_var2)
print(add_var)
```

```
60
```

In [58]:
```
my_var,my_var2=23,34 #assigning values to the variable in one go, make sure the no. of variables and input matches
```

In [59]:
```
print(my_var,my_var2)
```

```
23 34
```

In [62]:
```
print(my_var+my_var2)
```

```
57
```

exploring various data types:-

integers

In [63]:
```
a = 98
print(a)
type(a) #to check the data type of the variable
```

```
98
```

Out[63]:
```
int
```

In [82]:
```
b=56
print(b)
type(b) #to check the data type of the variable
```

```
56
```

Out[82]:
```
int
```

In [97]:
```
c1 = print(a*b)
```

```
5488
```

In [99]:
```
type(c1) #data type of print() is none type... That's why showing NoneType
```

Out[99]:
```
NoneType
```

floating points

In [86]:
```
c = 45.3
print(c)
type(c)
```

```
45.3
```

Out[86]:
```
float
```

In [87]:
```
d= 66.44
print(d)
type(d)
```

```
66.44
```

Out[87]:
```
float
```

In [101]:
```
e1=print(c*d)
type(e1)  #data type of print() is none type... That's why showing NoneType
```

```
3009.7319999999995
```

Out[101]:
```
NoneType
```

complex numbers

In [103]:
```
e= 0+1j
print(e)
type(e)
```

```
1j
```

Out[103]:
```
complex
```

In [110]:
```
f= 2+6j
print(f)
type(f)
```

```
(2+6j)
```

Out[110]:
```
complex
```

Lets learn about String Concatenation in Python

String Concatenation is the technique of combining two strings. String Concatenation can be done using many ways.

1) Using + operator

2) Using join() method

3) Using % operator

4) Using format() function

5) Using , (comma)

Method 1: String Concatenation using + Operator

In [114]:
```
# Defining strings
var1 = "Hello " #if you see after "o" we gave space
var2 = "World"

# + Operator is used to combine strings
var3 = var1 + var2 #maintain the space between the variable and operator
print(var3)
```

```
Hello World
```

In [120]:
```
var4= "Let's " #if you see after "s" we gave space
var5= "Do it"

var6 = var4 + var5
print(var6)
```

```
Let's Do it
```

In [124]:
```
var7= "25"
var8= "64"
# above are strings thus it will not return the sum
var9 = var7 + var8
print(var9)
```

```
2564
```

In [137]:
```
language = "Python"

print("Hi welcome to my class of " + language + " language") # allowed but rarely used in industry
```

```
Hi welcome to my class of Python language
```

In [128]:
```
var10="24"
var11=54

var12= var10 + var 11
print(var12)  #ERROR since both are of different data types
```

```
  File "C:\Users\abc\AppData\Local\Temp/ipykernel_15804/663606736.py", line 4
    var12= var10 + var 11
                       ^
SyntaxError: invalid syntax
```

Method 2: String Concatenation using join() Method

The join() method is a string method and returns a string in which the elements of the sequence have been joined by str separator.

This method combines the string that is stored in the var1 and var2. It accepts only the list as its argument and list size can be anything.

In [134]:
```
var1 = "Hello"
var2 = "World"

# join() method is used to combine the strings
print("".join([var1, var2])) #here no space(" ") seperator is available

# join() method is used here to combine
# the string with a separator Space(" ")
var13 = " ".join([var1, var2])

print(var13)
```

```
HelloWorld
Hello World
```

In [133]:
```
var14 = "try"
var15 = "method 2"

var16 = " ".join([var14, var15])

print(var16)
```

```
try method 2
```

Method 3: String Concatenation using % Operator

We can use the % operator for string formatting, it can also be used for string concatenation.

It's useful when we want to concatenate strings and perform simple formatting.

The %s denotes string data type. The value in both the variable is passed to the string %s and becomes "Hello World".

In [135]:
```
var1 = "Hello"
var2 = "World"

# % Operator is used here to combine the string
print("% s % s" % (var1, var2))
```

```
Hello World
```

In [136]:
```
var17 = "try"
var18 = "method 3"

print("%s %s" % (var17, var18))
```

```
try method 3
```

Method 4: String Concatenation using format() function

str.format() is one of the string formatting methods in Python, which allows multiple substitutions and value formatting.

It concatenate elements within a string through positional formatting. The curly braces {} are used to set the position of strings.

The first variable stores in the first curly braces and the second variable stores in the second curly braces. Finally, it prints the value “Hello World”.

In [139]:
```
#old way
language = "Python"
day1 = "Sunday"
day2 = "Monday"
print("Hi welcome to class of {} language, join every {} & {} at 8pm.".format(language,day1,day2))
```

```
Hi welcome to class of Python language, join every Sunday & Monday at 8pm.
```

In [145]:
```
#old way
language = "Python"
day1 = "Sunday"
day2 = "Monday"
print("Join every {} & {} at 8pm to learn {} language.".format(day1,day2,language)) #the order to variables can vary.
```

```
Join every Sunday & Monday at 8pm to learn Python language.
```

In [148]:
```
# current standard approach used in python.
language2 = "SQL"
print(f"Hi welcome to class of {language2} language")
```

```
Hi welcome to class of SQL language
```

In [149]:
```
language2 = "SQL"
day1 = "Sunday"
day2 = "Friday"
print(f"Join every {day1} & {day2} at 8pm to learn {language2} language.")
```

```
Join every Sunday & Friday at 8pm to learn SQL language.
```

In [151]:
```
a = 58
b = 92
c = a * b
print(f"the product of two nums a: {a} and b: {b} is = {c}")
```

```
the product of two nums a: 58 and b: 92 is = 5336
```

Boolean

In [156]:
```
stat1= True
stat2= False

print(stat1, stat2)
```

```
True False
```

In [159]:
```
print(stat1)
type(stat1)
```

```
True
```

Out[159]:
```
bool
```

In [160]:
```
print(stat2)
type(stat2)
```

```
False
```

Out[160]:
```
bool
```

In [163]:
```

num1 = 32
num2 = 21
comparision = num1 < num2 #n1 less than n2

print(f"the comparision is: {comparision}")
```

```
the comparision is: False
```

In [170]:
```
num1 = 32
num2 = 21
comparision = num1 != num2 #not equal to

print(f"the comparision is: {comparision}")
```

```
the comparision is: True
```

In [175]:
```
num1 = 32
num2 = 78
comparision = num1 > num2 #n1 greater than n2

print(f"the comparision is: {comparision}")
```

```
the comparision is: False
```

In [177]:
```
num1 = 32
num2 = 78
comparision = num1 == num2 # comparision, if n1 is equals to n2

print(f"the comparision is: {comparision}")
```

```
the comparision is: False
```

Method 5: String Concatenation using (, comma)

“,” is a great alternative to string concatenation using “+”. when you want to include single whitespace.

Use a comma when you want to combine data types with single whitespace in between.

In [178]:
```
var1 = "Hello"
var2 = "World"

# " , " to combine data types with a single whitespace.

print(var1, var2)
```

```
Hello World
```

In [181]:
```
var19 = "Use"
var20 = "Comma"

print(var19,var20)
```

```
Use Comma
```

thank you :)

Check out the next document for Day 2 topics.

In [ ]: