Basic of Python

`Print` Function

```
# Print Function can print strings, numbers, booleans, decimals etc.
print('hello')
Output --> hello

print('2')
Output --> 2

print(False)
Output --> False
```

```
# Print multiple values
print('India', 'aditya', 'honey')
Output --> India aditya honey

print('Hello', 5, True)
Output --> Hello 5 True
```

Why there is space in between output of multiple values.

- `sep=' '` in print function there is parameter called separator which by default have space in it. That's why we have space between multiple output.

```
# Let's say in place of space i want /
print('India', 'aditya', 'honey', sep='/')
Output --> India/aditya/honey
```

When printing multiple values but in diff print func. we get output in new line

- Its because of `\n` → jump to new line

```
# what if i want it in same line
print('hello', end=' ')
print('world')

Output --> hello world
```

Data Types

💡 Basic Types - (Integer, float, complex, boolean, and string)

Container Types - (List, Tuples, Sets and Dictionary)

User_defined Types - Class

```
# Integer
print(4)

# float
print(4.5)

# boolean
print(True)

# complex
print(4+5j)
```

```
# string
print('hello')

# list
print([1,2,3,4]) --> [1,2,3,4]

# Tuple
print((1,2,3,4)) --> (1,2,3,4)

# Sets
print({1,2,3,4}) --> {1,2,3,4}

# Dict
print({'Name':'adi', 'age': 30, 'gender':'male'})
--> {'Name': 'adi', 'age': 30, 'gender': 'male'}
```

Comments

A piece of which is not executed by interpreter or compiler

We use `#` for comment,

Variables

Containers for future use.

```
# here x is variable
x = 'hello'
print(x)
```

```
# Dynamic Typing
python understand the data type of variable by its own we
dont have assighn is data type.
```

```
#Syntax technique
a = 4; b = 5; c = 3
print(a)
print(b)
print(c)

--> 4
        5
        3

a,b,c = 4,5,6
print(a)
print(b)
print(c)
```

```
--> 4
        5
        6

a = b = c = 6
print(a)
print(b)
print(c)

--> 6
        6
        6
```

Keywords

- Python is a case sensitive programming language.
- A keyword is a word that is reserved by a program because the word has a special meaning. Keywords can be command or parameters. Every programming language has a set of keywords that cannot be used as variable names.

Identifiers

A python identifier is a name used to identify a variable, function, class, module or other object

- can only start with an alphabet or _
- Followed by 0 or more letter, _ and digits
- keywords cannot be used as an identifiers.

Taking user input

To take input from user we will use `input()` function.

```
# adding two number
first_num = input('enter first number: ')
second_num = input('enter second number: ')

#OUTPUT
enter first number: 20
enter second number: 24

result = first_num + second_num
print(result)

#output
2024 # not the answer what we expected
```

This happens because when we give value to input function, it always take them as string datatype.

Type Conversion

Converting one datatype into another.

This is not a permanent conversion.

```
first_num = int(input('enter first number: '))
second_num = int(input('enter second number: '))

#OUTPUT
enter first number: 20
enter second number: 24

result = first_num + second_num
print(result) --> 44
```

Literals

Literal is a raw data given in a variable

- Numeric
- String
- Boolean
- Special

```
# NUMERIC Literals
a = 0b1010 #Binary Literals
b = 100 #Decimal Literals
c = 0o310 # Octal Literal
d = 0x12c #Hexadecimal Literal

# Float Literal
float_1 = 10.5
float_2 = 1.5e2 # output --> 150.0
float_3 = 1.5e-3  # output --> 0.0015
```

```
# String Literals
single_line_string = 'hello'

multiline_string = """
                                                you can write anything here
                                                whatever you want
                                        """
emoji_type_string (unicode_character) = u"\U0001f600" #output --> 🙂
raw_string (if_printing_html_code) = r"raw \n string"
# output --> raw \n string. As you can see there-
# -will be no line break as we used \n
```

```
# Special Literal

a = None
```

```
print(a)

#Output
None
```

Operators

It is used to perform operations on variables and values.

- Arithmetic operators
- Comparison Operators
- Logical Operators
- Bitwise
- Assignment
- Identity
- Membership

Arithmetic - `+` , `-` , `/` , `*` , `//` , `%` , `**`

U want the quotient - (Integer Division)

```
x = 4.5
y = 2

a = x // y
print(a)

# --> 4.5 // 2 = 2 and this will only
# return int even if you give float number for division
```

U want reminder - (True Division)

```
x = 4.5
y = 2

a = x % y
print(a)

# --> 4.5 % 2 = 0.5
```

Logical

`AND` , `OR` , `NOT`

Bitwise - only work on binary

```
x = 2
y = 3

print (x & y)
print (x | y)

+-------------+-------------+
| bitwise '&' | bitwise '|' |
+-------------+-------------+
|      010    |     010     |
+-------------+-------------+
|      110    |     110     |
+-------------+-------------+
|      010    |     110     | --> Output = 2     3
+-------------+-------------+
```

Assignment

`=` , `+=`, `-=`, `*=`

Identity

```
a = 3
b = 3

print (a is b)
# True

a = [1,2,3]
b = [1,2,3]

print (a is b)
# False

# "is" tells if both values have the same memory location.
```

Membership

```
x = 'Delhi'

print('D' not in x)
# False
```

If-else Statements

This means conditions.

if

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

elif

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

else

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

LOOP

While

```
i = 1
while i < 6:
  print(i)
  i += 1
```

💡 **Note:** remember to increment i, or else the loop will continue forever.

Break Statement

stop the loop even if the while condition is true:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

```
#OUTPUT
1
2
3
```

Continue Statement

we can stop the current iteration, and continue with the next:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)

# OUTPUT
1
2
4
5
6
```

Else Statement

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

FOR

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

```
#Looping through string
for x in "banana":
  print(x)
```

`Range()` Function

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6):
  print(x)
```

💡 **Note:** that range(6) is not the values of 0 to 6, but the values 0 to 5.

```
#Increment the sequence with 3 (default is 1)
for x in range(2, 30, 3):
  print(x)
# OUTPUT
2
5
8
11
14
17
20
23
26
29

for x in range (10, 0, -1):
    print(x)

#OUTPUT
10
9
8
7
6
5
4
3
2
1
```

Else

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

NESTED LOOP

A nested loop is a loop inside a loop.

- The "inner loop" will be executed one time for each iteration of the "outer loop":

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)

#OUTPUT
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

```
# The Star Triangle
*
* *
* * *
* * * *
* * * * *

rows = int(input('Enter the no.')

for i in range(1, rows + 1):
        for j in range(0, i):
                print("*", end=" ")
```

Pass

`for` loops cannot be empty, but if you for some reason have a `for` loop with no content, put in the `pass` statement to avoid getting an error.

```
for x in [0, 1, 2]:
  pass
```