PYTHON FUNDAMENTALS:

[@kyoumah_07]

MODULE -2

★: PYTHON BASIC INPUT AND OUTPUT

- **PYTHON OUTPUT**

In python we have the print() function to print something to console or terminal

[example:

```
print("Hello world")
#output: Hello world
```

- **Parameters and Syntax of "print()" function**

parameters accepted by print() functions can vary but they are 5.

```
print(object = separator = end = file = flush=)
```

- object = values to be printed
- sep (optional) = allows to seperate multiple objects inside print().
- end (optional) = allows us to add specific values like new line "\n", tab "\t".
- file (optional) = where the values are printed. Default is sys.stdout (screen)
- flush (optional) = boolean specifying if the output is flushed or buffered. Default: false.

example:

```
print("Good Morning!", end = ' ')
print("Hello", "world", sep = '.')

#output: Good Morning! Hello.world
```

• PYTHON INPUT :→

→ python uses the input() function to take user input

→ input(prompt)

example:

```
num = input("Enter a number")
print('You Entered', num)
```

•→ we can explicitly define what kind of input we want from a user using type casting.

example:

```
num = int(input("Enter a number"))
```

⊛: TYPE-CASTING | CONVERSION

type-casting = process of changing | converting data of one type to another.

example: converting int → str

Python supports two types of casting | conversion.

- Implicit Conversion :→ automatic type conversion
- Explicit Conversion :→ manual type conversion

- Implicit Conversion :→ python automatically converts one data type to another.

example

```
a = 100
b = 101.10
c = a+b
print(c)

#output= 201.10
```

```
print(type(c))
#output: <class 'float'>


a = <int>
b = <float>

<int> + <float> = <float>
```

[Note:→ we cannot add str and int. It gives a TypeError. in such case explicit conversion are used.

- **Explicit-type Conversion:**

  → helps to convert the data type of an object to required data type.

  → functions like int(), float(), str() can be used to perform type casting.

[example:

```
a = "42"      # string
b = 0         # int
c = a + str(b)      # b typecasted to a string
print(c)
print(type(c))

#output: 420
         <class float>
```

syntax [x = typecasting-function(y)]

Note: Data loss may occur during typecasting ex→ float → int

```
a = 420.01
b = int(a)
print(b)

#output: 420
```

OPERATORS IN PYTHON

An operator is a symbol / function that performs some operations on operands.

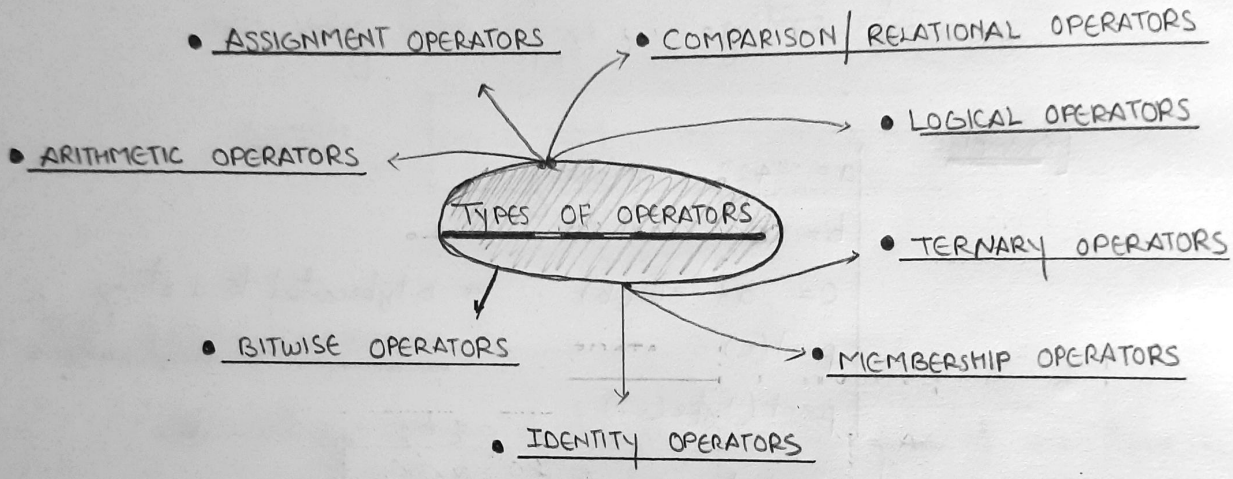
```
print(5+6)
#output 11
```

operands

5 + 6

operator

They allows us to perform operations on variables and values.



ARITHMETIC OPERATORS:

performs basic arithmetic operations

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (Remainder)
**	Exponentiation
//	floor Division

code:

```
x = 10
y = 3
print(x+y)
print(x-y)
print(x*y)
print(x/y)
print(x%y)
print(x**y)
print(x//y)
```

ASSIGNMENT OPERATORS

These operators are used to assign value to variables.

Operator	Description
=	Assign value
+=	add and assign
-=	subtract & assign
*=	multiply & assign
/=	divide & assign
%=	Modulus & assign
**=	Exponent & assign
//=	floor division & assign

RELATIONAL / COMPARISON OPERATORS

Operator	Description
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

- These are used to compare two values.
- Returns a boolean value either True or false.

LOGICAL OPERATORS

◎ logical operators are used to combine conditional statements.

Operators	Description
and	Logical AND → True if all values are true
or	Logical OR → True if either value is true
not	Logical NOT → negates values

◎ They also returns a boolean value

BITWISE OPERATORS

These operators work on bits and perform bit-by-bit operations.

Operators	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Left shift
>>	Right shift

code:

```
x = 10
y = 4
print(x&y)
print(x|y)
print(x^y)
print(~x)
print(x<<2)
print(x>>2)
```

output:

```
0
14
14
-11
40
2
```

example: 10 in binary 1010

4 in binary 0100

1010 & 0100 = 0000	1010 | 0100 = 1110	1010 ^ 0100 = 1110
0 in binary	14 in decimal	14 in decimal

~(00001010)

= (1)1110101

↓ negative,

(2's Complement representation)

00001010 + 1

(0)0001011

(1) = -11

Note:- Integers in python are represented using two's complement. Leftmost bit assigns its number val [0 = positive, 1 = negative].

⊙ This behaviour is consistent for an integer X:
⇒ ~x will always result in -(x+1).

- 1010 << 2 ⇒ left shift by 2 bits

⇒ shifting once

1 0 1 0 0

shifting again

1 0 1 0 0 0

→ [Converting 101000 to decimal
⇒ 40 in decimal.

→ shifting left bits n times will multiply the number 2^n times.

- x >> 2
1010 >> 2
⇒ 0010 converting to decimal = 2
⇒ shifting bits to right n times will divide the number 2^n times

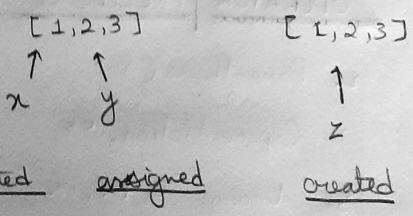
- IDENTITY OPERATORS

→ Used to check if two objects are in the same memory.

Operator	Description
is	True if both point to same object
is not	True if both do not point to same object

(example)

x = [1, 2, 3]
y = x
z = [1, 2, 3]



print(x is y) # True (y refers to x)
print(x is z) # false (x object does not refer to z)

- because lists are mutable in python. when we create a new list even though it has the same content as another list (say x = [1,2,3], z = [1,2,3]) its a distinct object in memory.
- when we assign y = x, it references to same object in memory.

- **MEMBERSHIP OPERATOR :→**

used to test whether a value or variable is present in a sequence (like a string, tuple, list)

Operator	Description	Example
in	True if value is found	'a' in 'apple' (True)
not in	True if the value is not found	'b' not in 'apple' (True)

```
x = [1, 2, 3, 4]
print(3 in x)        # True
print(5 not in x)    # false
```

- **TERNARY (CONDITIONAL) OPERATORS:**

→ python also supports a ternary operator for conditional assignments.

Syntax	Description
x if condition else y	a = 10 if x > 5 else 5

- **OPERATOR PRECEDENCE IN PYTHON** (Highest-to-lowest) [Associativity]

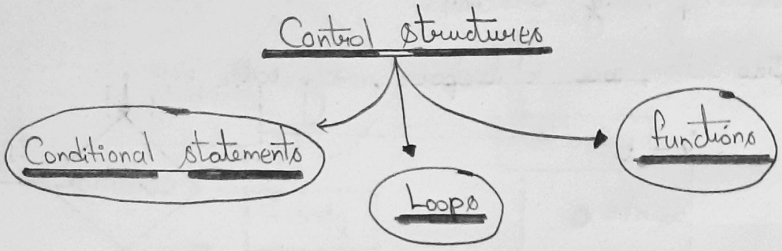
1. Parentheses ()
2. Exponentiation. **
3. Unary ~, +x, -x
4. Mul, Div, //, %
5. Addition, Substraction : +, −
6. Bitwise shift <<, >>
7. Bitwise AND, &
8. Bitwise XOR, OR : ^, |
9. Comparision [==, !=, etc.]
10. Identity and Membership
11. Logical NOT: not
12. Logical AND: and
13. Logical OR: or
14. Conditional statement
15. Assignment operator

PYTHON CONTROL-FLOW:

CONTROL STRUCTURES:

In programming control structures are constructs or instructions that govern the flow of execution of a program.

- Determines the order in which statements are executed, how many times a particular piece of code is executed, based on condition or events.



CONDITIONAL / DECISION CONTROL STATEMENTS

- → They allow us to execute different block of codes based on certain conditions.
- → Conditional statements in python are "if", "elif", "else".

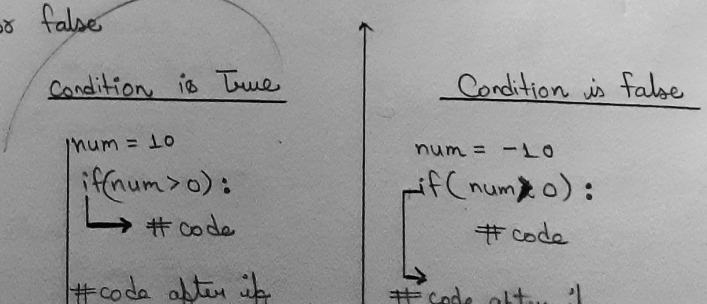
If statement

- An if statement executes a block of code only when the specified condition is met.

Syntax:

```
if condition:
    #body of if statement
```

- The condition is a boolean expression, ex (num > 10) that evaluates to either True or false



example:

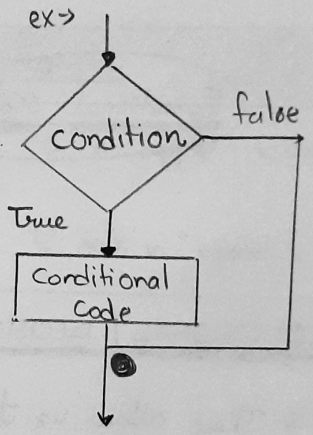
```
num = int(input("Enter a number:"))

if num > 0:
    print(f"{num} is positive.")

print("Hello_world")
```

- **INDENTATION IN PYTHON**
  - python uses identation to define a block of code, and organize it instead of using brackets.
  - Identation are whitespaces, any code ex->

```
x = 1

count = 0

if (x != 0):
    count += x

print(count)
```

Identation → ..... (4 spaces)

if condition code inside if block.



- **If-else statement**
  - if can have optional else close. The else statement executes if the condition in the if statements evaluates to false.

Syntax

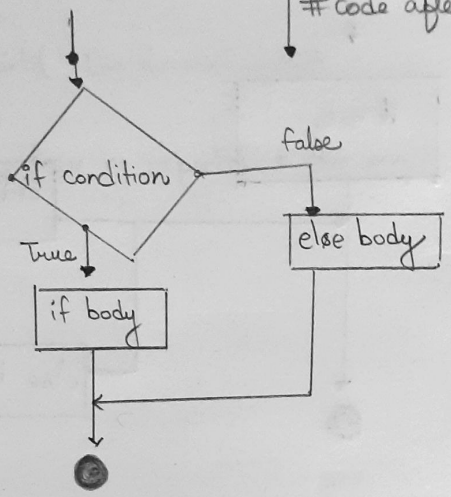
```
if Condition:
    # body of if statement
else:
    # body of else statement
```

Condition is True

```
number = 10
if number > 0:
    #code
else:
    #code
#code after
```

Condition is false

```
number = 10
if number > 0:
    #code
else:
    #code
#code after
```



example:

```
num = int(input("Enter a number"))
if(num%2==0):
    print("Even")
else:
    print("odd")
```

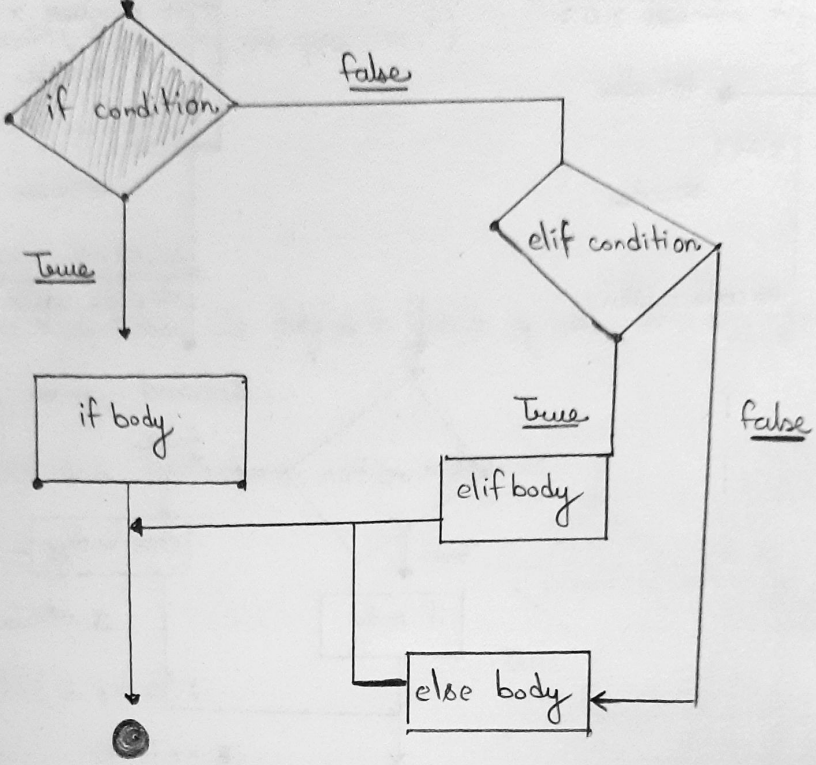
- **if-elif...else statement**

- whenever we need to make a choice between more than two alternatives, we can use the if... elif... else statement.

syntax:

```
if condition1:
    #code block1
elif condition 2:
    #code block2
else:
    #code block3
```

- control-flow of [if... elif... else].



[Example:]

```
num = int( input( " Enter the Number"))
if num > 0:
        print ( "Positive")
elif num < 0:
        print( "Negative")
else :
        print( " Zero")
```

- [NESTED - If statements :→

  if statements inside an if-statement are called nested if statements.

example:

```
grade = int(input("Enter your Grade"))
attendance = int(input("Enter your attendance"))

if grade >= 95:
    if attendance >= 90:
        print("Good Job")
    else:
        print("Can do much better")
else:
    print("we have no expectations from you")
```

} → nested-if [if inside if]

PYTHON LOOPS

- loops allows us to repeat a block of code multiple times. Python provides two type of loops
  - for loop
  - while loop

- **FOR-LOOP:**

  A for loop is used to iterate over a sequence (like a list, tuple, string, or range) and perform action on each element.

  Syntax

```
for variable in sequence:
    # body of loop
```

example:

- **looping through a sequence**

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

- **looping through a string**

```
x = "Hello_world"
for i in x:
    print(i)
```

- **looping through Python range()**

```
for i in range(5):
    print(i)
```

Note: range() takes 3 parameter. range(start, stop-1, increment/decrement)

- we can also use conditional [if-else] statements or other loops inside a loop.
- Looping inside a loop is called nested looping.

- **WHILE - LOOP**

A while loop continues to execute a block of code as long as a given condition is [True].

Syntax:

```
while condition:
    # body of loop
```

example:

```
i = 1
while i <= 5:
    print(i)
    i += 1          ← incrementing value of i per loop.
```

- **NESTED - LOOPS:**

→ A for loop can have number of for loops inside it, for each cycle of the outer loop, the inner loop completes its entire sequence of iterations.

example

```
for i in range(2):
    for j in range(2):
        print( i, " ", j )
```

#output

```
0   0
0   1
```

LOOP CONTROL STATEMENTS

- python provides statements that control the flow of loops:

  break : Terminates the loop prematurely.

  continue : Skips the current iteration and continues with next iterations.

  pass : Does nothing and is used as a placeholder.

- **Break**

  break keyword in python used within loops to immediately terminate the loop and resumes execution at the next statement after the loop.

ex:-

```
for i in range(1,6):
    if i ==3:
        break        #exits loop when i=3
    print(i)
```

- **Continue:**

  continue is a keyword used within loops to skip the rest of the current iteration and move onto the next iteration.

ex:-

```
for i in range(1,6):
    if i==3:
        continue
    print(i)
```

- **pass:**

  pass is keyword used within loops, functions and methods etc. as a simple placeholder and it does nothing.

ex:-

```
for i in range(5):
    if i ==2:
        pass
    print(i)
```

DIFFERENCE BETWEEN THE WHILE & FOR LOOP

- [WHILE LOOP]:
  - Loops runs as long as the condition is True.
  - Possible to create an infinite loop if condition never becomes false.
  - Need to handle the increment or update of the loop variable manually within the loop block.
  - Can be used with range().

- [FOR-LOOP]
  - Iterates over elements in an iterable one by one until the iterable is exhausted.
  - No-risk of creating an infinite loop.
  - No need to handle iteration control manually.
  - Primarily used with range() to control the number of iterations.

COMMENTS IN PYTHON

- used to add explanation or notes to the code, comments are ignored by the interpreter during execution.
- useful for code documentation

1. Single-line comments: → starts with #
2. Multiple (multi-line comments): → python inbuilt does not have support and uses docstrings, using ["""] or '''[]. however they are stored as string objects in memory when outside function or class definition, hence better avoid them whenever possible.