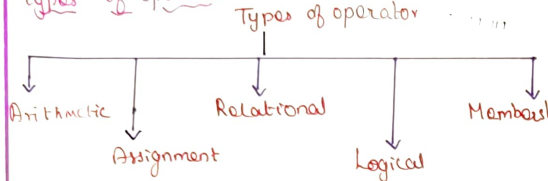
OPERATORS IN PYTHON

OPERATOR:

* a symbol that tells the compiler to perform specific mathematical or logical functions.

Types of Operator



Arithmetic operator: a = 17 , b = 2

Operator	Description	Example	Output
+	Addition	print (a+b)	19
−	Subtraction	print (a−b)	15
*	Multiplication	print (a* b)	32
/	Division	print (a/b)	8
%	Modulus ↳returns remainder	print (a % b)	1
//	Floor Division ↳returns whole number quotient	print (a//b)	8
**	Exponent ↳power	print (a** b)	289

Logical operator: AND, OR , NOT

A	B	A AND B	A OR B	NOT A
F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	F

Assignment Operators:

Operator	Description	Examples
=	Assigns values from right to left	a = 17
+=	Add AND	c += a ⇒ c = c+a
−=	Subtract AND	c −= a ⇒ c = c−a
*=	Multiply AND	c* = a ⇒ c = c*a
/=	Divide AND	c /= a ⇒ c = c/a
%=	Modulus AND	c %= a ⇒ c = c%a
**=	Exponent AND	c** = a ⇒ c = c**a
//=	Floor Division	c //= a ⇒ c = c//a

Relational Operator : a = 5 , b = 2

Operator	Description	Example	Output
==	equal to	print (a==b)	False
>	Greater than	print (a>b)	True
<	Less than	print (a<b)	False
>=	greater than equal to	print (a>=b)	False
<=	less than equal to	print (a<=b)	False
!=	Not equal to	print (a!=b)	True

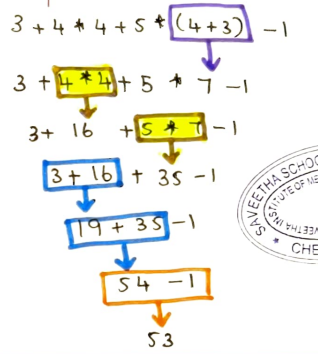
Membership Operator:

* Operator used to validate the membership of a value.
  Eg: a = [5, 1, 8, 7]
* Types
1. in operator ⇒ print (8 in a)
   output : True
2. not in operator ⇒ print (0 not in a)
   output : True

OPERATOR PRECEDENCE

Parenthesis
Power
Division
Multiplication
Addition
subtraction
Left to Right

Example:



COMMENTS IN PYTHON

* not executed by compiler
* used for documentation of code.

Example

```
# This is a comment
print ("Hello, world!")
"""
This is a comment
written in more than just one line
"""
print ("Hello, World")
```

TOPIC: CONDITIONAL STATEMENTS

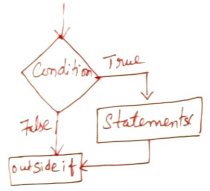
* Performs different computations depending on specific boolean constraint (True or False)

Conditional - if

* used to test a condition
* if the condition is true statements inside if will be executed.

Syntax:

```
if (condition 1):
    statement 1
    statement 2
    :
    statement 3
```



Example:
Program to provide flat rs.500, if the purchase amount is greater than 2000

```
a=int(input("Enter the purchase amt"))
if(a >= 2000):
    a = a-500
print ("Amt to pay", a)
```

O/P

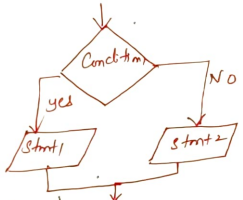
```
Enter the purchase amount: 2500
Amt to pay : 2000
```

if ... else

* Used to test a condition, when the alternative is present.
* if the condition is true, statements inside the if gets executed otherwise statements inside else part gets executed.

Syntax:

```
if (condition1):
    statement 1
else:
    statement 2
```



Example:
program to find the given number is odd or even

```
n=int(input("Enter the number"))
if (n %2 ==0):
    print ("even number")
else:
    print ("odd number")
```

O/P

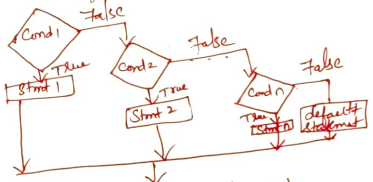
```
Enter a number : 4
even number
```


if ... elif .. else

* Used to check more than one condition.
* If condition1 is false, it checks the condition 2 of the elif block. If all the conditions are false, then the else part is executed.

Syntax:

```
if (condition 1):
    statement1
elif (condition 2):
    statement 2
elif (condition 3):
    statement 3
else: default statement
```



Example: Student mark system

```
mark=int(input("enter ur mark"))
if (mark >=90):
    print("grade : S")
elif (mark >=80):
    print ("grade : A")
elif (mark >=70):
    print ("grade : B")
elif (mark >=50):
    print ("grade : C")
else: print ("fail")
```

O/P

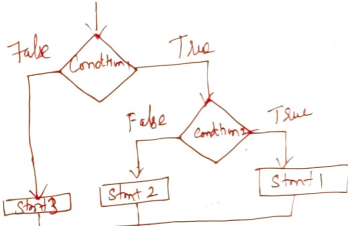
```
Enter ur mark: 78
grade : B
```

Nested if .. else

* Any number of condition can be nested inside one another
* If condition1 is true, it checks another if condition2. If both the conditions are true statement1 get executed otherwise statement 2 gets executed.

Syntax:

```
if (condition):
    if (condition 1):
        statement 1
    else:
        statement 2
else:
    statement 3
```



Example: Greatest of 3 numbers

```
a = input("Enter the value of a")
b = input("Enter the value of b")
c = input("Enter the value of c")
if (a>b):
    if(a>c):
        print("The greatest",a)
    else:
        print("The greatest",c)
else:
    if(b>c):
        print("The greatest",b)
    else:print("The greatest",c)
```

O/P

```
Enter the value of a: 5
Enter the value of b: 1
Enter the value of c: 8
The greatest 8
```

LOOPING STATEMENTS

Def: allows to execute a statement or group of statements multiple times

FOR LOOP

* used to iterate over a sequence (list, tuple, string)
* Loop continues until the last element in the sequence is reached.

Syntax:

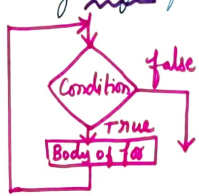
```
for i in sequence :
    print(i)
```

String	List	Tuple
Eg: for i in "Rama": print(i)	Eg: for i in [2, 3, 5, 6, 9]: print(i)	Eg: for i in (2, 3, 1): print(i)
o/p: R a m a	o/p: 2 3 5 6 9	o/p: 2 3 1

* Sequence of numbers can be generated using range() function.

Syntax:

```
for i in range(start, stop, steps):
    body of for loop
```



Example Program: Prime or not

```
n = int(input("Enter a number"))
for i in range(2, n, 1):
    if(n%i == 0):
        print("The num is not a prime")
        break
    else:
        print("The num is a prime number")
        break
```

o/p

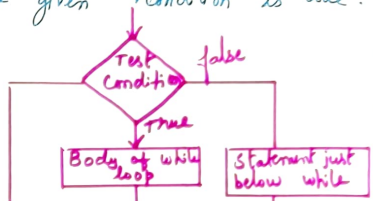
```
Enter a number 7
The num is a prime number
```


WHILE LOOP

* used to repeatedly execute set of statements as long as the given condition is true.

Syntax:

```
initial value
while(condition):
    body of while loop
    increment.
```



Example Program: Sum of n numbers

```
n = int(input("Enter n"))
i = 1
Sum = 0
while (i<=n):
    Sum = Sum + i
    i = i + 1
print(Sum)
```

o/p: Enter n 10
55

o/p: Enter n 5
15

Iteration	Variable	i<=num	Body of the loop
1	num = 5, i = 1	True	Sum = 1
2	num = 5, i = 2	True	Sum = 1+2 = 3
3	num = 5, i = 3	True	Sum = 3+3 = 6
4	num = 5, i = 4	True	Sum = 6+4 = 10
5	num = 5, i = 5	True	Sum = 10+5 = 15
6	num = 5, i = 6	False	exit the loop

Break

* It terminates the current loop and executes the remaining statement outside the loop

Example:

```
for i in "welcome":
    if (i == "c"):
        break
    print(i)
```

o/p

```
w
e
l
```

Continue

* It terminates the current iteration and transfers the control to the rest iteration in the loop.

Example:

```
for i in "welcome":
    if (i == "c"):
        continue
    print(i)
```

o/p

```
w
e
l
o
m
e
```

FUNCTION

Function is a group of related statements that performs a specific task

Types of function

- **Built-in function (or) Pre-defined function** [Library functions]
  - Eg: len() function
    ```
    x = [1, 2, 3, 4, 5]
    print(len(x))
    ```
    ↳ returns length of list
- **User-defined functions** [Defined by the programmer to reduce the complexity of big problems]
  - Eg:
    ```
    x = 3
    y = 4
    def add():
        print(x+y)
    add()
    ```

Types of Parameters:

Positional parameter | Keyword Parameter | Default parameter | Variable length parameter

1. Positional parameter

* Number of parameter in the function definition should match exactly with number of arguements in the function call.

Eg:
```
def student(name, roll):
    print(name, roll)
student("Ram", 98)
```
o/p: Ram 98

2. Keyword parameter

* During the function call, the calling function identifies the parameters by the function's parameter name.
* order of the arguments can be changed.

Eg:
```
def student(name, roll, mark):
    print(name, roll, mark)
student(mark=90, roll=11078, name="Ram")
```
o/p: Ram, 11078, 90

3. Default parameter

* If the function is called without the argument, the argument gets its default value in function definition.

Eg:
```
def student(name, age=17):
    print(name, age)
student("Kumar")
student("ajay")
```
o/p: Kumar 17, ajay 17

4. Variable length parameter

* If the number of arguments to be passed, is not known in advance, asterisk (*) can be used before the parameter name to denote the variable length of parameter.

Eg:
```
def student(name, *mark):
    print(name, mark)
student("Ram", 98, 88)
```
o/p: Ram (98, 88)

Local and Global Scope

Scope: Refers to the place where it is declared, used and can be modified. It is the lifetime of the variable in the program.

Local scope:
* variable created inside a function belongs to the local scope of that function.

Global scope:
* Variable with global scope can be used anywhere in the program.
* variable defined outside the function.

Eg:
```
a = 50          → Global Variable
def add():
    b = 20      → Local Variable
    c = a+b
    print(c)
```

Return Values:

"return" Keyword is used to return the values from the function.

Eg:
- return a - return 1 variable
- return a,b - return 2 variable
- return a+b - return expression
- return 8 - return value.

Fruitful function:
* function that returns a value

```
def add():
    a = 10
    b = 20
    c = a+b
    return c
c = add()
print(c)
```
o/p: 30

Void function:
* Function that performs action, but don't returns any value

```
def add():
    a = 10
    b = 20
    c = a+b
    print(c)
add()
```
o/p: 30

Function Composition

* Ability of a function to call from within another function.
* Result of each function is passed as the argument of next function.
* output of one function is given as input of another.

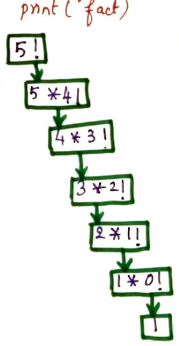
```
def add(a,b):
    c = a+b
    return c
def mul(c,d):
    e = c*d
    return e
c = add(10,20)
e = mul(c,30)
print(e)
```
o/p: 900

Recursion

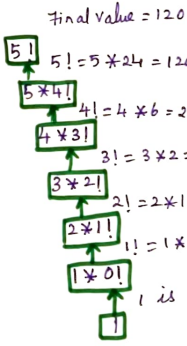
* A function calling itself till it reaches the base value (stop point) of function call.

Eg: Factorial of n

```
def fact(n):
    if(n==1):
        return 1
    else:
        return n*fact(n-1)
n = int(input("Enter the number"))
fact = fact(n)
print(fact)
```
o/p: Enter the number 5, 120



Final value = 120



STRING

Def: Sequence of characters represented in quotation marks single quotes, double quotes
* immutable → contents of the string cannot be changed after creation.
* Python will get the input at runtime by default as a string.

Operations on string

1. Indexing:
* Individual characters in a string is accessed using a index.
* Index must be an integer (positive or negative), and starts from 0 to n-1

String A	H	E	L	L	O
positive index	0	1	2	3	4
negative index	-5	-4	-3	-2	-1

Eg: a = "HELLO"
print(a[0])
o/p: H

Eg: print(a[-1])
o/p: O

* Access the string from beginning
* Access the string from end

2. Slicing:
* Extracting substring from a string.
* operator [start : stop] (or) [start : stop : steps]

Eg:
a = "HELLO"
print[0:4] o/p - HELL
print[:3] - HEL
print[0:] - HELLO

3. Concatenation:
* operator '+' joins the text on both sides of operator

Eg: a = "saree"
b = "earth"
print(a+b)
o/p: sareearth

4. Repetition
* operator '*' repeats the string on the left hand side for the number of times given on the right hand side

Eg: a = "saveetha"
print(2*a)
o/p: saveethasaveetha

5. Membership
* "in" operator check a particular character in string.
* "not in" operator check character is not in string

Eg: S = "good morning"
"m" in S → o/p: True
"a" not in S → o/p: True

Built-in-methods: a = "happy birthday"

1) a.capitalize() o/p 'Happy Birthday'
2) a.upper() o/p 'HAPPY BIRTHDAY'
3) a.lower() o/p 'happy birthday'
4) a.title() o/p 'Happy Birthday'
5) a.swapcase() o/p 'HAPPY BIRTHDAY'
6) a.split() o/p ['happy', 'birthday']
7) a.count('substring') a.count('happy') o/p 1
8) a.replace(old,new) a.replace('happy', 'wish you happy') o/p wish you happy birthday
9) a.join(b) b = 'happy' a = "-" a.join(b) o/p = 'h-a-p-p-y'
10) a.isalpha() o/p = False
11) a.isdigit() o/p = False
12) a.startswith('substring') a.startswith("h") o/p True
13) a.endswith('substring') a.endswith('y') o/p True
14) a.find('substring') → returns index if found, returns -1 if not found. a.find("happy") o/p 0
15) len(a) o/p 14
16) min(a) o/p ' '
17) max(a) o/p 'y'

LIST

Def: Ordered sequences of items that can be different data types
* Values in the list are called elements / items.
* Notation: [ ]
* Mutable → elements in the list can be changed.

Operations on list

1. Indexing:
Eg: a = [2,3,4,5,6,7,8,9,10]
print(a[0]) o/p 2
print(a[-1]) o/p 10

2. Slicing:
Eg: print(a[0:3]) o/p [2,3,4]

3. concatenation:
Eg: b = [20,30]
print(a+b) o/p [2,3,4,5,6,7,8,9,10,20,30]

4. Repetition:
Eg: print(b*3) o/p [20,30,20,30,20,30]

5. Membership:
Eg: 5 in a o/p True
100 in a o/p False
2 not in a o/p False

6. Updating:
Eg: a[2] = 100
print(a)
o/p [2,3,100,5,6,7,8,9,10]

7. comparison:
Eg: b = [2,3,4]
a == b o/p False
a != b o/p True

Built-in-methods: a = [1,2,3,4,5]

1) a.append(element) Eg: a.append(6) print(a) o/p: [1,2,3,4,5,6]
2) a.insert(index, element) Eg: a.insert(0,0) print(a) o/p: [0,1,2,3,4,5,6]
3) a.extend(b) Eg: b = [7,8,9] a.extend(b) print(a) o/p [0,1,2,3,4,5,6,7,8,9]
4) a.sort() Eg: a.sort() print(a) o/p [0,1,2,3,4,5,6,7,8]
5) a.index(element) Eg: a.index(8) o/p: 8
6) a.reverse() Eg: a.reverse() print(a) o/p: [8,7,6,5,4,3,2,1,0]
7) a.remove(element) Eg: a.remove(1) print(a) [8,7,6,5,4,3,2,0]
8) a.pop() Eg: a.pop() o/p 0
9) a.pop(index) Eg: a.pop(0) o/p 8
10) a.count(element) Eg: a.count(6) o/p 1
11) a.copy() Eg: b = a.copy() print(b) o/p [7,6,5,4,3,2]
12) len(list) Eg: len(a) o/p: 6
13) min(list) Eg: min(a) o/p: 2
14) max(list) Eg: max(a) o/p: 7
15) a.clear() Eg: a.clear() print(a) o/p: []
16) del(a) Eg: del(a) print(a) o/p: Error: name 'a' is not defined