Python OOP

Content

- Data Type
    - List, Tuples, Dictionary, String
    - Operators
- Control Statement
    - Decision Making Statement
    - Looping Statement
- Function
    - Built-In Function
    - User-Defined Function
- Classes and Object
    - Calling through object
    - Calling through class by passing object as argument
- Constructors and Destruction
    - Default Constructor
    - Parameterized Constructor
- Inheritance
    - Single Inheritance
    - Multiple Inheritance
    - Multilevel Inheritance
    - Hierarchical Inheritance
    - Hybrid Inheritance
- Polymorphism
    - Function overloading
    - Operator overloading
    - Function overriding
- File

- Exception Handling
- Pandas
- NumPy

Computer Programming

Computing programming is the process of designing and building an executable computer program for accomplishing a specific computer task. Programming is the art and science of translating a set of ideas into a program – a list of instruction a computer can follow. The person writing a program is known as a programmer (also a coder). The exact form the instructions take depend on the programing Language used.

Different Types of translators

Complier

Compiler is a translator which is used to convert program in high-level language to low-level language. It translates the entire program and also reports the errors in source program encountered during the translation.

Interpreter

Interpreter is a translator which is used to convert programs in high-level language to low-level language. Interpreter translates line by line and reports the error once it encountered during the translation process.

Python

Python is introduced in 1991 by Guido van Rossum, a Dutch computer programmer.

The language places strong emphasis on code reliability and simplicity so that the programmers can develop rapidly. It's small, very closely resembles that English language.

Features of Python

1. Interpreter

   Python is processed at runtime by the interpreter executes the code line by line at a time. This makes debagging easy and thus suitable for beginners. Internally, python converts the source code into an intermediate form called bytecodes which is then translated into native language of specific computer to run it. No need to worry about linking and loading with libraries, etcetera.
2. Interactive

   You can use a Python prompt and interact with the interpreter directly to write your programs.
3. Object-Oriented

   Python support object-oriented technique of programming.
4. Beginner's Language

   Python is a great language for the beginner-level programming and supports the development of a wide range of Applications.
5. Expressive Language

   Python language is more expressive means that it is more understandable and readable.
6. Cross-Platform Language

   Python can run equally an different platforms such as Windows, Linux, Unix and Macintosh etcetera. So, we can say that Python is a portable language.
7. Free and open-source

   Python language is freely available at official web address. This source-code is also available. Therefore, it is open source

PEP 8

PEP 8 in Python style guide. It's a set of rules for how to format your Python code to maximize its readability. Writing code to a specification helps to make large code bases with lots of writers, more uniform and predictable, too. PEP is actually an acronym that stands for Python Enhancement Proposal. They also provide a reference point (and a standard) for the pythonic way to write code. The PEP8 in the style Guide for Python Code, and it covers:

- Formatting
- Comments
- Naming conventions

It also has a lot of programming recommendations and useful tips on various topics, which aim to improve readability (and reliability) of your code.

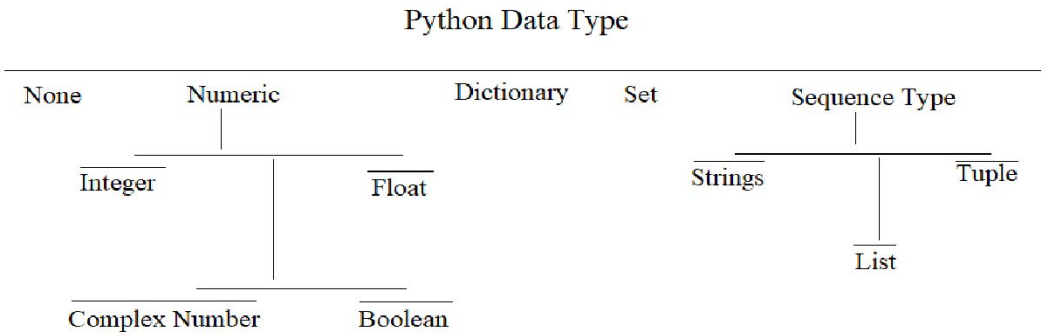
Data Types

Python is strongly, dynamically typed.

- Strong Typing
  Means that the type of a value doesn't change in unexpected ways. A string containing only digits doesn't magically become a number, as may happen in Perl. Every change of type requires an explicit conversion.
- Dynamic Typing
  Means that runtime objects (values) have a type, as opposed to static typing where variable have a type.

Note: Weak Typing means allowing access to the underlying representation. In C, you can create a pointer to characters, then tell the complier. You want to use it as a pointer to integer.

Data Types



Data Types Falls under two different types

1. Immutable : The data that can't be change over time

   Number, Strings, Tuples

2. Mutable : The data that can be change over time

   Lists, Dictionary, Sets

List

List is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type

Creating a list is as simple as putting different comma-separated values square brackets.

For example

Code: my_list = [“Badal”,” Wahid”, 2,001]

Code: my_list = [1, 2, 3, 4, 5]

Type of Lists

Empty list

Code: My_empty_list = []

List of Integer

Code: my_in_list = [1, 2, 3, 4, 5]

List of mix data type

Code: my_ix_list = [1, “Hello”, 3.4]

Nested list

Code: my_nested_list = [“mouse', 1][2]

Accessing the Values

To access values in list, use the square brackets for slicing along with the index or indices to obtain values available at that index

Indexing of elements are as follow:

Values:	Badal,	Aman,	102,	94
Positive:	0	1	2	3
Negative:	-4	-3	-2	-1

Code: my_list = [“Badal”,” Aman”, 2,001]

Code: print(“Printing first element”, my_list[1])

Output: Aman

Code: print(“Element 1 to 4”, my_list[1:4])

Output: Element 1 to 4 [Badal, Wahid, 2, 001]

Printing all Values

Code: print(“Printing all element : “, my_list[:])

Output: Printing all element : [Badal, Wahid, 2, 001]

Updating list

Code: my_list_new['Physics', 'Chemistry', 'Math', 1998, 2000, 2002]

Code: print(my_list_new)

Output: ['Physics', 'Chemistry', 'Math', 1998, 2000, 2002]

Code: my_list_new [1] = 'Computer'

Code: print(“Updated 1 positional element : “, my_list)

Output: Updated 1 positional element : ['Physics', 'Computer', 'Math', 1998, 2000, 2002]

Code: my_list[3:5] = [1992, 1994, 1996]

Output:['Physics', 'Computer', 'Math', 1992, 1994, 1996]

Deleting Element

Code: del my_list[0]

Code: print("After deleting first element : ", my_list

Output: After deleting first element : [Wahid, 2, 001]

Deleting Entire list

Code: del my_list[]

Code: print(my_list[])

Output: Traceback (most recent call last):

File "<stdin>", line, in <module>

Name Error: name 'my_list is not defined

Slicing

we can slice lists by accessing range of items in a lists by using the slicing operation colon. Slicing can be done by two methods. 1.) Positive slicing 2.) Negative slicing

Positive Slicing

Code: my_list0 = ['C', 'Python', 'C++', 1, 2]

Code: print(my_list0[0:])

Output: ['C', 'Python', C++', 1, 2]

Code: print(my_list0[1:5])

Output: ['Python', 'C++', 1, 2]

It will start form 1 but it will end before 5 so if you want to print 5 you have to type 6 to get desire result

or output

Negative Slicing

Code: print(my_list0[:-4])

Output: ['C']

Code: print(my_list0[:])

Output: ['C', 'Python', 'C++, 1, 2]

Basic list operation

# Concatenate	# Repitation	# Membership
[1, 2, 3] + [4, 5, 6]	[Hi!] *4	3 in [1, 2, 3]
[1, 2, 3, 4, 5, 6]	[Hi! Hi! Hi! Hi!]	True

List function

append()

" To insert new item in the list "

```
Code: list1 = [ 'C', 'C++', 'HTML']
      list1.append("python")
      print(list1)
```

Output: ['C', 'C++', 'HTML', 'Python']

index()

" To find the index of the element "

Code: print(list1.index('C'))

Output: 0

extend()

" Iterates over its argument and adding each element to the list and extending the list. The length of the list increases by number of elements in its argument. "

```
Code: list01 = [ 'C', 'C++', 'HTML', 'Python']
      list02 = [1998, 2000, 2002, 2004]
      list01.extend(list02)
      print(list01)
```

Output: ['C', 'C++', 'HTML', 'Python', 1998, 2000, 2002, 2004]

insert()

" To insert item at specific location "

```
Code: list01.insert(0, 'Binary')
      : print(list01)
```

Output: ['Binary', 'C', 'C++', 'HTML', 'Python', 1998, 2000, 2002, 2004]

" Here 0 is the index number and 'Binary is the data to be insert."

count

" To count the occurrence of the same data.

It is case sensitive. "

```
Code: print(list01.count('Python'))
```

Output: 1

pop()

" To delete element by its index number. "

```
Code: print(list01.pop(2))
      print(list01)
```

Output: 'C++'

['Binary', 'C', 'HTML', 'Python', 1998, 2000, 2002, 2004]

reverse()

" To reverse the list "

Code: list01.reverse()

print(list01)

Output: [2004, 2002, 2000, 1998, 'Python', 'HTML', 'C', 'Binary']

reversing using slicing

Code: print(list01[::-1])

Output: ['Binary', 'C', 'HTML', 'Python', 1998, 2000, 2002, 2004]

sorting

" Sorting can be only done at one data type at one time. "

Code: sorted(list01, reverse=True) # for Desecnding

Output: Traceback (most recent call last):

File "<pyshell#8>", line 1, in <module>

sorted(list_pro, reverse = True)

TypeError: '<' not supported between instances of 'int' and 'str

Code: list03 = ['Python', 'HTML', 'C', 'Binary']

sorted(list03, reverse=False) # for Acending

print(list03)

Output: ['Binary', 'C', 'HTML', 'Python']

len()

" len for sorted.

The list is sorted based on the length of each element from lowest count to highest.

Data should be either int or str. "

Code: list03.sort(key = len)

print(list03)

Output: ['C', 'HTML', 'Python', 'Binary']

copy()

" To copy list "

```
Code: import copy
        newlist = copy.copy(old_lit)
```

clear()

" To clear all the items of the list. "

```
Code: newlist.clear()
```

" To clear all items. "

max()

" To find the maximum. "

```
Code: listno = [0, 1, 2, 3, 4]
        print(max(listno))
```

```
Output: 4
```

min()

" To find minimum. "

```
Code: print(min(listno))
```

```
Output: 0
```

Tuple

A Tuple is a collection of objects which ordered and immutable. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples uses parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses.

Example :

```
tup1 = ('Physics', 'Chemistry', 1998, 2000)
tup2 = (1, 2, 3, 4, 5)
tup3 = "a", "b", "c", "d"
```

Accessing value in tuple is just like lists. You can use indexing to access items of

tuple. For illustration.

Code: tup1 = ('Physics', 'Chemistry', 'Math', 1920)

print(tup1[0:])

Output: ('Physics', 'Chemistry', 'Math', 1920)

Function in Tuple

converting list into tuple

Code: list1 = ['Science', 1998]

tuple1 = tuple(list1)

print(tuple1)

Output: ('Science', 1998)

Here are the function which are similar as in list.

max(tuple)

min(tuple)

index(tuple)

count(tuple)

len(tuple)

sum(tuple)

The function which are not valid in tuple.

copy()

sort()

append()

extend()

insert()

remove()

pop()

reverse()

Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionary are written with curly brackets, and they have keys and values.

Example :

```
car = {
        "brand" :   "Ford",
        "model"     "Mustang",
        "year" :    1964
      }
```

Keys | Values

Keys + Values = Pair

```
Code: car = {
        "brand"     :     "Ford",
        "model"     :     "Mustang",
        "year"      :     1964
              }
        print(car)
Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Accessing Items

you can access the items of a dictionary by referring to its key name, inside square brackets.

```
Code: a = car ['mode'l]
```

```
print(a)
```

Output: Mustang

Type of dictionary

```
# Empty Dict
my_dict = {}
# Dict with integer
my_int_dict = {1: 'apple', 2: 'Orange'}
# Dict with mixed keys
my_mix_dict = { 'name': 'wahid', 1:[2,4,8]}
```

Updating dict

```
Code: car['year'] =2018
print(car)
```

Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 2018}

```
# Loop through a Dict
Code: for x in car:
        print(x)
```

Output: brand

model

year

```
Code: for x in car:
        print(car[x])
```

Output: Ford

Mustang

2018

```
# for loop with functions
# values()
Code: for x in car.values():
```

```
        print(x)
```

Output: Ford

```
        Mustang
        2018
```

items()

```
Code: for x, y in car.items():
        print(x,y)
        or
    for x in car.items():
        print(x)
```

Output: ('brand', 'Ford')

```
        ('model', 'Mustang')
        ('year', 2018)
```

Some basics function

get()

" to get the value of key. "

```
Code: print(car.get('brand'))
```

Output: Ford

conditional

```
Code: if "model" in car:
        print('yes ' ,model,' is one of the keys in the car dict')
```

Output: yes, 'model' is one of the keys in the car dict

Deleting

```
Code: car.pop("model")
    print(car)
```

Output: {'brand': 'Ford', 'year': 2018}

popitem()

" The popitem() method removes the last inserted item (in version before 3.7, a random item is removed instead)"

```
Code: car.popitem()
        print(car)
Output: {'brand': 'Ford'}
```

del()

" del() keyword removes the item with specified key name. "

```
Code: car = {
        "brand"     :     "Ford",
        "model"     :     "Mustang",
        "year"      :     1964
                }
        print(car)
Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
Code: del car1['model']
        print(car1)
Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

clear()

" clear() used to empty the dict

```
Code: car1.clear()
        print(car1)
Output: {}
```

copy dict

" you cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes mode in dict1 will automatically also be mode in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary

method copy()"

Syntax:

```
new_dict = old_dict.copy()
```

Another way to make copy is to use the built-in function dict()

Syntax:

```
new_dict=dict(old_dict)
```

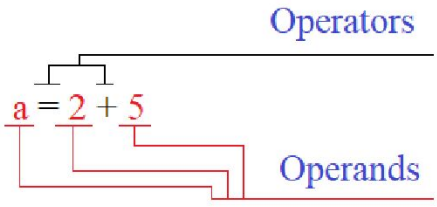
fromkey

" To convert set into dictionary. "

```
Code: one = ('name', 'age', 'class')
      dict2 = dict.fromkeys(one)
      print(dict2)
Output: {'name': None, 'age': None, 'class': None}
```

Operators

Operators are special symbols Python that carry out arithmetic or logical computation. The value that the operator operators on is called the operand.



Example

```
>>> 2 + 3
5
```

Here, + is the operator that performs addition. 2 and 3 are the operands and 5 is the

output of the operation.

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etcetera

Table for Arithmetic Operators

If x = 15, y = 4

Operator	Meaning	Example	Result
+	Add two or more operands or unary plus	x + y	19
-	Subtract right operand from left or unary minus	x - y	11
*	Multiply two or more operands	x * y	60
/	Divide left operand by right one (results in float)	x / y	3.75
%	Modulus remainder of division	x % y	3
//	Floor division that results whole number	x // y	3
**	Exponent- left operand raised to power of right	X ** y	50625

Comparison Operators

Comparison operators are used to compare value. It returns either True or False accordingly to the condition.

Table for Comparison Operators

If x = 10, y = 12

Operator	Meaning	Example	Result
>	Greater than – True if left operand is greater than right	x > y	False
<	Less than – True if left operand is less than the right	x < y	True
==	Equal to – True if both operands are equal	x == y	False
!=	Not equal to – True if operands are not equal	x != y	True
>=	Greater than or equal to – True if left operand is greater than or equal to the right	x >= y	False
<=	Less than or equal to – True if left operand is less than or equal to the right	x <= y	True

Logical Operators

Logical operators are the and operators

If x = True, y = False

Operator	Meaning	Example	Result
and	True if both the operands are true	x and y	False
or	True if either of the operands is true	x or y	True
not	True if operands is false, complements the operand		

Truth Table

And

and will result into True only if both operands are true.

A	B	A and B
1	1	1
1	0	0
0	1	0
0	0	0

Or

Or will result into True if any of the operands is True.

A	B	A or B
1	1	1
1	0	1
0	1	1
0	0	0

Not

Not operator is used to invert the truth value

A	not A
1	0
0	1

Bitwise Operators

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

For example:

In the following table:

Let x = 10 (0000 1010) and y = 4 (000 0100)

Operator	Meaning	Example	Result
&	Bitwise AND	x & y = 0	0(0000 0000)
|	Bitwise OR	x | y = 14	14(0000 1110)
~	Bitwise NOT	~ x = - 11	11(1111 0101)
^	Bitwise XOR	x ^ y =14	14(0000 0101)
>>	Bitwise right shift	x >> 2 = 2	2(0000 0010)
<<	Bitwise left shift	x >> 2 = 4	40(0010 1000)

Assignment Operators

Assignment operators are used in python to assign values to variables.

a = 15 is a simple assignment operator that assigns the 5 on the right to the variable "a" on the left.

There are various compound operators in python like a + = 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5

Assignment Table

Operator	Example	Equivalent to
=	x = 5	x =5
+=	x + = 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x % = 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x ** = 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5

^=	x ^= 5	x = x ^ 5

Special Operators

Python language offers some special types of operators like the identity operator or the membership operator. They are described below.

Identity operators

"Is" and "is not" are the identity operators. They are used to check if two values (or variables) are located on some part of the memory. Two variables that are equal does not imply that they are identical.

Operator	**Meaning**	**Example**
is	True if the operands are identical (refers to the same object)	x is True
not	True if the operands are not identical (do not refer to the same object)	x is not True

Membership Operators

"in" and "not in" are the membership operators in Python. They are used to test whether a value or variable is found in a sequence.

String, list, tuple, set and dictionary. In a dictionary we can only test for presence of key, not the value.

Operator	**Meaning**	**Example**	**Result**
in	True if value/variable is found in sequence	5 in x	True
not in	True if value/variable is not found in sequence	5 not in x	False

Keywords

Keywords are the reserved words in Python. We cannot use a keyword or a variable name, function name or any other identifiers.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
or	def	from	nonlocal	while
assert	del	global	not	with

async	elif	if	or	yield

Note: To print all the keyword

```
Import keyword
Print(keyword.kwlist)
```

Control Statement

Decision Making

Looping Statement

Decision Making statement

Decision making is required when we want to execute a code only if a certain condition is satisfied.

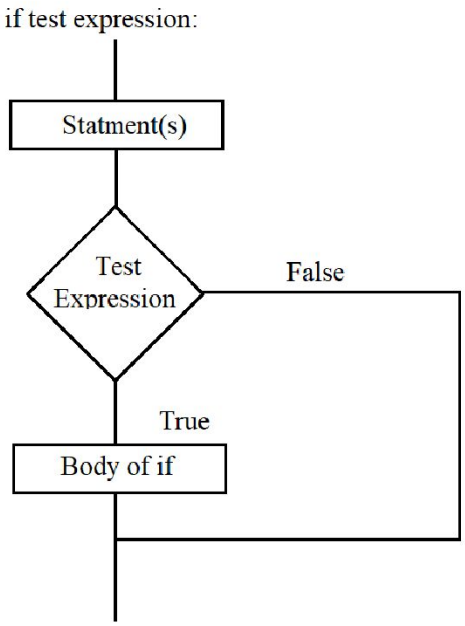
In Python three are three kind of Decision making method

1.) If

2.) Elif

3.) Nested if

Syntax for If



Example:

```
Code: # if the number is positive, we print an appropriate message.
        num = 3
        if num > 0:
                print(num, " is a positive number")
        print("This is always printed)
```

```
Output: 3 is a positive number:
        This is always printed
```

Example:

```
Code: num1 = int(input("Enter first number: ")
      num2 = int(input("Enter second number: ")
      if num1 < num2:
            print("{} is less than {}".format(num1,num2))
      else:
            print("{} is less then {}".format(num2,num1))
Output: Enter first number: 12
        Enter second number: 21
        12 is less than 24
```

Elif

```
Code: marks = int(input("Enter student's percentage: ")
      if marks <= 33:
      print("Failed")
      elif marks > 33 and marks < 45:
            print("E Grade")
      elif marks > 45 and marks < 65:
            print("D Grade")
      elif marks > 65 and marks < 75:
            print("C Grade")
      elif marks > 75 and marks < 80:
            print("B Grade")
      elif marks > 80 and marks < 90:
            print("A Grade")
```

```
        elif marks > 90 and marks < 95:
                print("A Grade)
        elif marks > 95 and marks < 99:
                print("S Grade")
        elif marks == 100:
                print("SSS+ Grade)
        else:
                print("Invalid Input!")
        print("Computer Generated Result")
```

Output: Enter student's percentage: 100

SSS+ Grade

Computer Generated Result

If else ladder

```
Code: name = input("Enter your name please: ")
        age = int(input("Enter your age please: ")
        print("Name: {} and Age: {}".format(name,age))
        detail_check = int(input("If your details are correct press 1 else press 2: "))
        if detail_check == 1:
                print("Accepted")
                if age >=17 and age < 21:
                        print("you can join.")
                else:
                        print("You are not eligible")
        else:
                print("Fill your detail again")
```

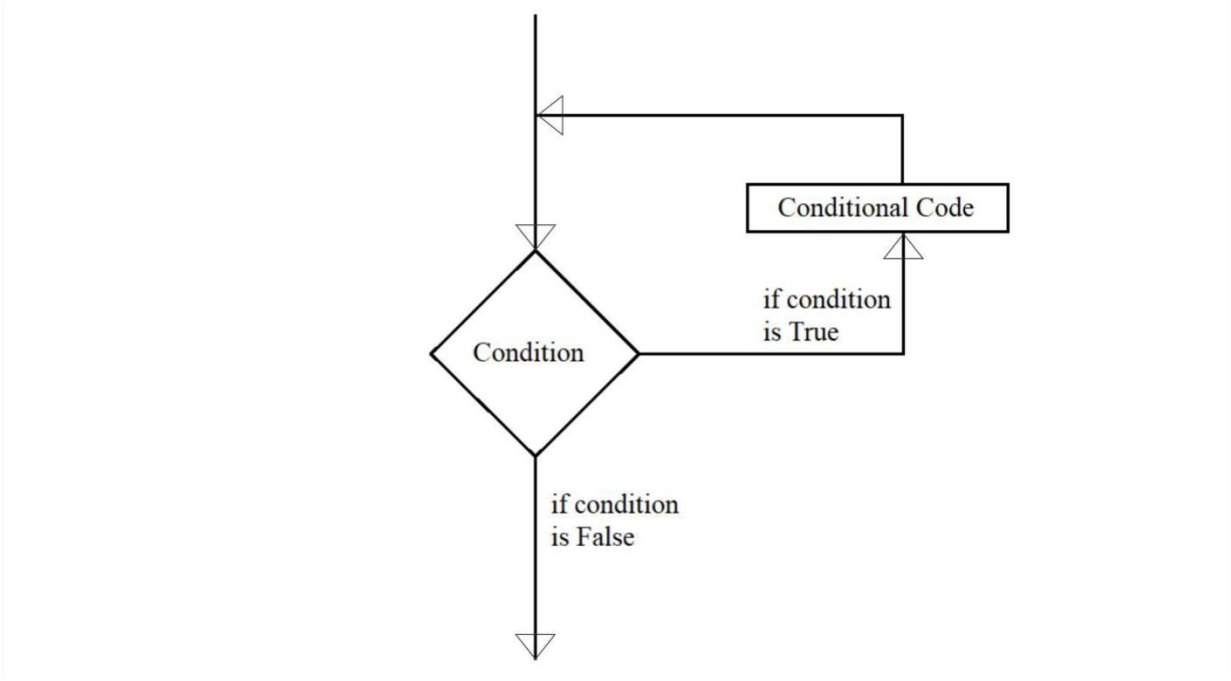
Code: Enter your name: Wahid Ali

Enter your age: 21

```
Name: Wahid Ali and Age: 21
If details are correct press 1 else press 2: 1
Accepted
You can join
```

Looping Statement

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times. Programming languages provides various control structures that allow for more complicated execution paths. Looping statement are the statements execute one or more statement repeatedly several number of times.



The above flowchart diagram illustrates a loop statement.

Python programming language provides different types of loops to handle looping requirement given follow:

1. while loop

   Repeats a statement or group of statements while a given condition is True. It tests the condition before execution the loop body.

2. for loop

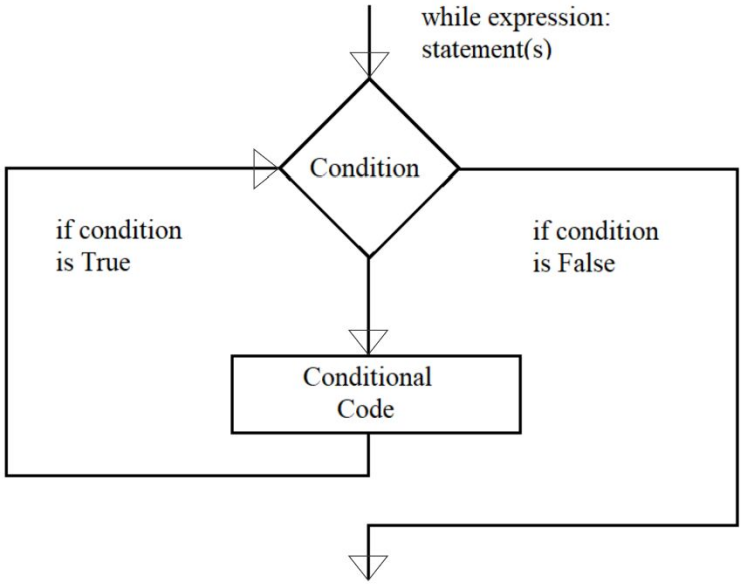
   Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

3. Nested loop

   You can use one or more loop inside any other while of for loop

while loop statements

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true. The syntax of a while loop in python is as follow. Here, statement(s) may be a single statement or a block of statement with uniform indent. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.



Example:

```
Code: count = 0
      While (count < 10):
            print("The count is: ", count)
            count = count + 1
      print("Good Bye!")
Output: The count is:  0
        The count is:  1
        The count is:  2
```

```
The count is:  3
The count is:  4
The count is:  5
The count is:  6
The count is:  7
The count is:  8
The count is:  9
```

The black here, consisting of the print and increment statement, is executed repeatedly until count is no longer less than 9, with each iteration, the current value of the index count is displayed and then increased by 1.

The infinite loop

A loop becomes infinite loop if a condition never becomes False. You must be cautions when using while loops because of the possibility that this condition never ends. Such a loop is called an infinite loop. An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

Example:

```
Code: var =1
        while var == 1 # This constucts an infinite loop
                num = int(input("Enter a number: "))
                print("you entered: ", num)
        print("Program ended Successfully:")
Output: Enter a number: 1
        You entered:  1
        Enter a number: 2
        You entered:  2
        Enter a number: 2
        You entered:  2
        Enter a number: 2
```

```
You entered:  2
Enter a number: 4
You entered:  4
Enter a number: ctrl+c
Traceback (most recent call last):
File "<pyshell#15>", line 2, in <module>
num = int(input("Enter a number: "))
KeyboardInterrupt
```

Note: The above example goes in an infinite loop and you need to use Ctrl + C to exit the program.

Using else Statement with loop.

Python supports having an else statement associated with a loop statement. If the else statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list.

Example:

Code:

```
count = 0
        while count < 5:
                print(count, “, is less than 5 “)
                count = count + 1
        else:
                print(count, “ is not less than 5”)
```

Output:

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
```

Single statement suits / one line while similar to the if statement syntax, if while clause consists only of a single statement, it may be placed on the same line as the while header.

Example:

```
flag = 1
While(flag): print('given flag is really true!")
print("good bye!")
```

The above example goes in an infinite loop and you need to use ctrl + f2 as ctrl + c to exit the program.

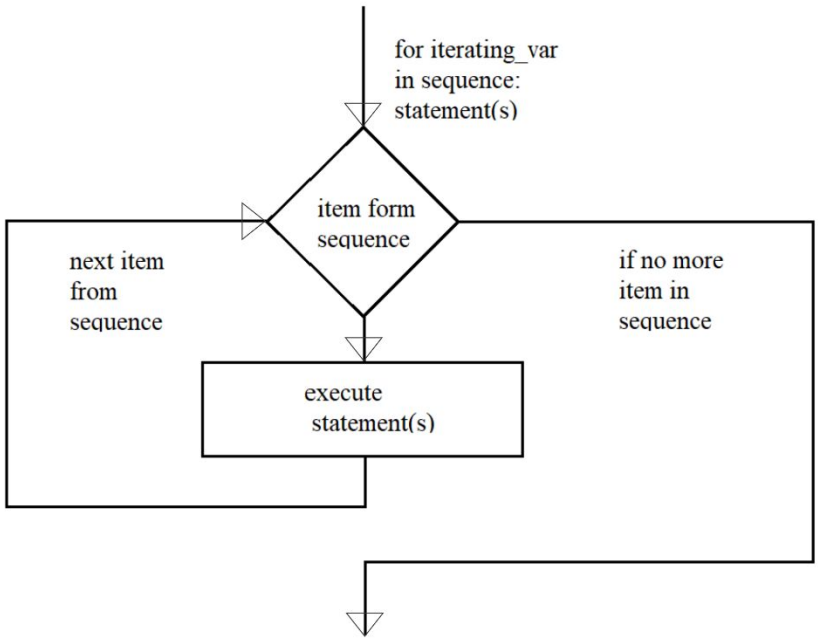
For loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.

Iterating over a sequence is called traversal.

Syntax

```
For var in sequence:
    Body of for statement
```



Here, var is the variable that takes the value of the item inside. The sequence each iteration. Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rent of the code using indentation.

Example:

```
Code: primes = [2, 3, 5, 7]
        For prime in primes:
                Print(prime)
Output: 2
        3
        5
        7
```

Example:

```
Code: for letter in 'Python': # traversal of a string sequence
                print("Current letter: ", letter)
        print()
Output: Current letter:  P
        Current letter:  y
        Current letter:  t
        Current letter:  h
        Current letter:  o
        Current letter:  n
```

Example

```
Code: # program to find the sum of all numbers stored in a list
        Numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
        sum = 0 # variable to store the sum
        # iterate over the list
        for var in numbers:
```

```
        print("The sum is", sum)
```

Output: The sum is 6

The sum is 11

The sum is 14

The sum is 22

The sum is 26

The sum is 28

The sum is 33

The sum is 37

The sum is 48

range() function

We can generate a sequence of number using range() function. Range(10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step size as range(start, stop, step size). Step size defaults to 1 if not provided. The built-in function range() is the right function to iterate over a sequence of numbers. It generates an iterator of arithmetic progressions.

Code: list(range(5))

Output: [0, 1, 2, 3, 4]

Example

Range() generates an iterator to progress integers string with 0 upto n-1. To obtain a list objects of the sequence, it is type casted to list(). Now this list can be iterated using the for statement.

Code: for var in list(range(5)):

```
        Print(var)
```

Output: 0

1

2

3

4

Example:

```
Code: for x in range(3, 6)
          print(x)
Output: 3
        4
        5

# iterating by sequence Index
Fruits = [‘banana’, ‘apple]
```