Python

Python is a general-purpose, dynamically typed, high-level, compiled and interpreted, garbage-collected, and purely object-oriented programming language that supports procedural, object-oriented, and functional programming.

Features of python:

1. **Easy to use and Read:** Python's syntax is clear and easy to read, making it an ideal language for both beginners and experienced programmers. This simplicity can lead to faster development and reduce the chances of errors.

2. **Object-Oriented:** Python has all features of an object-oriented language such as inheritance, method overriding, objects, etc. Thus it supports all the paradigms and has corresponding functions in their libraries. It also supports the implementation of multiple inheritances.

3. **Portable:** A programming language is portable if it allows us to code once and runs anywhere. This means the platform where it has been coded and where it is going to run need not be the same. This feature allows one of the most valuable features of object-oriented languages-reusability.

4. **Dynamically Typed** - The data types of variables are determined during run-time. We do not need to specify the data type of a variable during writing codes.

5. **High-level** - High-level language means human readable code.

6. **Compiled and Interpreted** - Python code first gets compiled into bytecode, and then interpreted line by line. When we download the Python in our system form org we download the default implement of Python known as CPython. CPython is considered to be Complied and Interpreted both.

7. **Garbage Collected** - Memory allocation and de-allocation are automatically managed. Programmers do not specifically need to manage the memory.

8. **Cross-platform Compatibility** - Python can be easily installed on Windows, Mac OS, and various Linux distributions, allowing developers to create software that runs across different operating systems.

9. **Rich Standard Library** - Python comes with several standard libraries that provide ready-to-use modules and functions for various tasks, ranging from web development and data manipulation to machine learning and networking.

10. **Open Source** - Python is an open-source, cost-free programming language. It is utilized in several sectors and disciplines as a result.

Data types in python:

Python Data Types are used to define the type of a variable. It defines what type of data we are going to store in a variable. The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.

The following are the standard or built-in data types in Python:

- **Numeric**
- **Sequence Type**
- **Boolean**
- **Set**
- **Dictionary**

Numeric Data Types in Python

The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number. These values are defined as Python int, Python float, and Python complex classes in Python.

Note – type() function is used to determine the type of Python data type.

Example:

```
a = 5
print("Type of a: ", type(a))
```

```
b = 5.0
print("\nType of b: ", type(b))

c = 2 + 4j
print("\nType of c: ", type(c))
```

Output:

```
Type of a:  <class 'int'>
Type of b:  <class 'float'>
Type of c:  <class 'complex'>
```

2. Sequence Data Types in Python

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion. There are several sequence data types of Python:

1. Python String
2. Python List
3. Python Tuple

1) String Data Type

A string is a collection of one or more characters put in a single quote, double-quote, or triple-quote. In Python, there is no character data type Python, a character is a string of length one. It is represented by str class.

Creating String

Strings in Python can be created using single quotes, double quotes, or even triple quotes.

Example:

```
String1 = 'Welcome to the Geeks World'
print(String1)
print(type(String1))
```

2) List Data Type

A list is a collection of different kinds of values or items. Python lists are mutable, so, we can change their elements after forming. The comma (,) and the square brackets [enclose the List's items] serve as separators.

Creating a List in Python
Lists in Python can be created by just placing the sequence inside the square brackets [].

Example:

```
List = ["Geeks", "For", "Geeks"]
print(List)
```

3) Tuple Data Type

Just like a list, a tuple is also an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by a tuple class.

Example:

```
Tuple1 = ('Geeks', 'For')
print(Tuple1)
```

3. **Python Boolean:** type is one of the built-in data types provided by Python, which represents one of the two values i.e. True or False. Generally, it is used to represent the truth values of the expressions.

Example

Input: 1==1
Output: True

Input: 2<1
Output: False

List:

A list is a ordered collection of different kinds of values or items. Python lists are mutable, so, we can change their elements after forming. The comma (,) and the square brackets [enclose the List's items] serve as separators.

Characteristics of Lists

The characteristics of the List are as follows:

- The lists are in order.
- The list element can be accessed via the index.
- Python Lists are mutable.
- The elements of various type can be stored in a list.

Creating a List

```
# Creating a list
my_list = [1, 2, 3, 4, 5]
print(my_list)  # Output: [1, 2, 3, 4, 5]
```

Accessing Elements

In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list. The index must be an integer. Nested lists are accessed using nested indexing.

```
# Accessing elements
print(my_list[0])  # Output: 1 (first element)
print(my_list[2])  # Output: 3 (third element)
print(my_list[-1])  # Output: 5 (last element)
print(my_list[-3])  # Output: 3 (3rd last element)
```

Modifying Elements

```
# Modifying elements
my_list[1] = 10
print(my_list)  # Output: [1, 10, 3, 4, 5]
```

Adding Elements

```
# Adding elements
my_list.append(6)
print(my_list)  # Output: [1, 10, 3, 4, 5, 6]

my_list.insert(1, 15)
print(my_list)  # Output: [1, 15, 10, 3, 4, 5, 6]
```

Removing Elements

```
# Removing elements
my_list.remove(10)
print(my_list)  # Output: [1, 15, 3, 4, 5, 6]

popped_element = my_list.pop()
print(popped_element)  # Output: 6 (last element)
print(my_list)  # Output: [1, 15, 3, 4, 5]
```

Slicing

```
# Slicing
sub_list = my_list[1:4]
print(sub_list)  # Output: [15, 3, 4]
```

Looping Through a List

```
# Looping through a list
for element in my_list:
    print(element)
```

Checking Membership

```
# Checking membership
print(3 in my_list)  # Output: True
print(10 in my_list)  # Output: False
```

Finding Length

```
# Finding length
print(len(my_list))  # Output: 5
```

Reversing a List

A list can be reversed by using the **reverse() method in Python**.

```
# Reversing a list
mylist = [1, 2, 3, 4, 5, 'Geek', 'Python']
mylist.reverse()
print(mylist)
```

Output

```
['Python', 'Geek', 5, 4, 3, 2, 1]
```

Dictionaries:

A Python dictionary is a data structure that stores the value in key:value pairs. Dictionaries in Python are a collection of key-value pairs where each key is unique and is used to store and retrieve values.

Creating a Dictionary

You can create a dictionary using curly braces {} with key-value pairs separated by colons, or by using the dict() function.

Using curly braces:

```
my_dict = {
    'name': 'Alice',
    'age': 25,
    'city': 'New York'
}
```

Using the dict() function:

```
my_dict = dict(name='Alice', age=25, city='New York')
```

Accessing Values

You can access values in a dictionary by using the keys.

```
print(my_dict['name'])  # Output: Alice
print(my_dict.get('age'))  # Output: 25
```

The **get** method is useful as it returns **None** (or a default value if specified) if the key is not found, instead of raising an error.

Adding or Updating Entries:

You can add new key-value pairs or update existing ones by simply assigning a value to a key.

```
my_dict['email'] = 'alice@example.com'  # Adding a new key-value pair
my_dict['age'] = 26  # Updating an existing key-value pair
```

Removing Entries

You can remove entries using the **del** statement or the **pop** method.

```
del my_dict['city']  # Removes the entry with key 'city'

# The pop method removes the key and returns its value
age = my_dict.pop('age')
print(age)  # Output: 26
```

Iterating Through a Dictionary

You can iterate through keys, values, or key-value pairs.

```
# Iterating through keys

for key in my_dict:
    print(key)

# Iterating through values

for value in my_dict.values():
    print(value)

# Iterating through key-value pairs

for key, value in my_dict.items():
    print(f'{key}: {value}')
```

Dictionary Methods

Here are some useful dictionary methods:

- **clear**(): Removes all items from the dictionary.
- **copy**(): Returns a shallow copy of the dictionary.

- **keys**(): Returns a view object with all the keys.
- **values**(): Returns a view object with all the values.
- **items**(): Returns a view object with all the key-value pairs.
- **update**(): Updates the dictionary with elements from another dictionary or an iterable of key-value pairs.

Example Usage

```
person = {
    'first_name': 'John',
    'last_name': 'Doe',
    'age': 30
}

# Accessing a value
print(person['first_name'])  # Output: John

# Adding a new key-value pair
person['email'] = 'john.doe@example.com'

# Updating a value
person['age'] = 31

# Removing a key-value pair
del person['last_name']

# Iterating through the dictionary
for key, value in person.items():
    print(f'{key}: {value}')

# Output:
# first_name: John
# age: 31
# email: john.doe@example.com
```

loops in Python:

Loops are used to repeatedly execute a block of code as long as a condition is met. We can run a single statement or set of statements repeatedly using a loop command.

There are two primary types of loops in Python:

1) For loop
2) While loop

For loop

Python's for loop is designed to repeatedly execute a code block while iterating through a list, tuple, dictionary, or other iterable objects of Python. The process of traversing a sequence is known as iteration.

Syntax of the for Loop

for iterator_var **in sequence:**
    { code to execute }

Loop iterates until the final item of the sequence are reached.

Example:

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

This will output:

```
0
1
2
3
4
```

While Loop in Python

In Python, a while loop is used to execute a block of statements repeatedly until a given condition is satisfied. When the condition becomes false, the line immediately after the loop in the program is executed.

Python While Loop Syntax:

```
while expression:
   statement(s)
```

Example:

```
count = 0
while count < 5:
   print(count)
   count += 1
```

This will output:

```
0
1
2
3
4
```

Example:

```
counter = 0
while counter < 10: # giving the condition
    counter = counter + 3
    print("Python Loops")
```

Output:

```
Python Loops
Python Loops
Python Loops
Python Loops
```

Nested Loops

Loops can be nested within other loops. The inner loop will be executed one time for each iteration of the outer loop.

Example:

```
for i in range(3):

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

This will output:

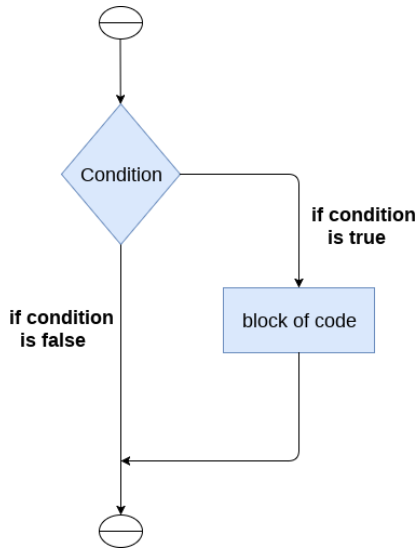
```
0 0
0 1
1 0
1 1
2 0
2 1
```

Conditional Statements:

1) If
2) If-else

if statement

The **if** statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.



The syntax of the if-statement is given below.

if expression:
{
#statement (s)
}

Example 1

```
1. # Simple Python program to understand the if statement
2. num = int(input("enter the number:"))
3. # Here, we are taking an integer num and taking input dynamically
4. if num%2 == 0:
5. # Here, we are checking the condition. If the condition is true, we will enter the blo
   ck
6.     print("The Given number is an even number")
```

Output:

```
enter the number: 10
The Given number is an even number
```

Example 2 : Program to print the largest of the three numbers.

```
1. # Simple Python Program to print the largest of the three numbers.
2. a = int (input("Enter a: "));
3. b = int (input("Enter b: "));
4. c = int (input("Enter c: "));
5. if a>b and a>c:
6. # Here, we are checking the condition. If the condition is true, we will enter the blo
   ck
7.     print ("From the above three numbers given a is largest");
8. if b>a and b>c:
9. # Here, we are checking the condition. If the condition is true, we will enter the blo
   ck
10.    print ("From the above three numbers given b is largest");
11.if c>a and c>b:
12.# Here, we are checking the condition. If the condition is true, we will enter the blo
   ck
13.    print ("From the above three numbers given c is largest");
```

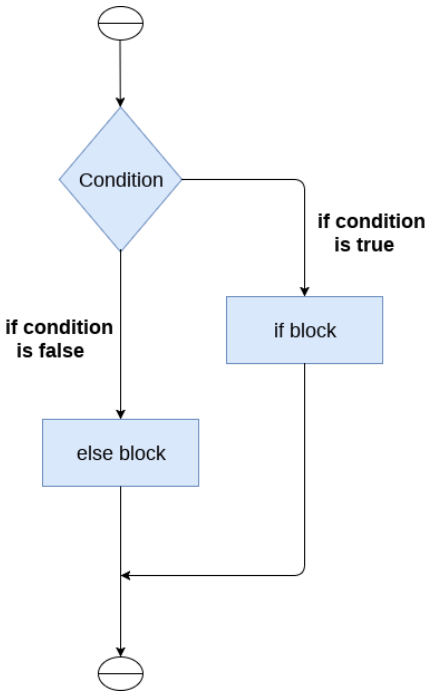
Output:

```
Enter a: 100
Enter b: 120
Enter c: 130
From the above three numbers given c is largest
```

The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.



The syntax of the if-else statement is given below.

```
if condition:
    #block of statements
else:
    #another block of statements (else-block)
```

Example 1 : Program to check whether a person is eligible to vote or not.

```
# Simple Python Program to check whether a person is eligible to vote or not.
age = int (input("Enter your age: "))
# Here, we are taking an integer num and taking input dynamically
if age>=18:
# Here, we are checking the condition. If the condition is true, we will enter the blo
ck
    print("You are eligible to vote !!");
else:
    print("Sorry! you have to wait !!");
```

Output:

```
Enter your age: 90
You are eligible to vote !!
```

Example 2: Program to check whether a number is even or not.

```
1. # Simple Python Program to check whether a number is even or not.
2. num = int(input("enter the number:"))
3. # Here, we are taking an integer num and taking input dynamically
4. if num%2 == 0:
5. # Here, we are checking the condition. If the condition is true, we will enter the blo
   ck
6.     print("The Given number is an even number")
7. else:
8.     print("The Given Number is an odd number")
```

Output:

```
enter the number: 10
The Given number is even number
```

These notes are prepared by **Suhail Abass Hurrah** (S.P College Srinagar, batch 2019)
For any queries, you can email me at Hurrahsuhail1@gmail.com