

Python Basics:

Syntax rules in python:

Python is case sensitive

Following indentation

Variables store data values.

Math modules are used to perform mathematical functions like, square root, exponential, power, trigonometry etc

Numeric Functions:

Integers, float, complex numbers

ceil(x) - returns integer close to x

fabs(x) - returns absolute value of x

Power and logarithmic functions:

pow(x,y) - returns x to the power of y

sqrt(x) - to find square root of x

exp(x) - find x to the power of e, where e = 2.718281

log(x, base) - returns log of x to the given base. By default, base is e

log2(x) - returns log of x to the base 2

log10(x) - return log of x to the base 10

Trigonometric and Angular functions:

Cos, sin, tan, asin, acos,

Degrees, radians

Data Types:

String - Collection of alphabets, words or other characters. Strings are arrays

List - Stores multiple items in a single variable of different types. It is mutable and ordered. []

Tuple - Stores multiple items in a single variable. It is immutable. ()

Set - Stores multiple items in a single variable. It is unordered. Items can be removed and added. {}

Dictionary - Stores values as key-value pairs. It is ordered and mutable. {key: value}

Arrays - Stores multiple values of same type

Operators:

Arithmetic: Used on numerical values to perform math operations. Addition, Subtraction, Multiplication, Division, Modulo, Exponent, Floor Division

Comparison Operator: Compares 2 values. ==, !=, >, <, >=, <=

Assignment Operator: Assign values to variables. =, +=, -=, *=, /=, %=, //=

Logical: To combine conditional statements. and, or, not

Bitwise: Used to compare numbers. bitwise AND, OR, NOT, XOR, Left shift, right shift [&, |, ~, ^, <<, >>]

Membership: To check if the sequence is present in the object. In, NOT In
Identity: Is, Is NOT

Decision Making:

If - Evaluates a condition. If condition is true, then the code the body gets executed else skipped

If else - If condition is true, if block is executed. If condition is false, else block is executed

Nested if else: If statements inside another if block

Control Structures:

For loops - iterates over a sequence like list, set, tuple, string, dictionary

While loop - executes a set of functions till the condition is true

Break - stops the loop even if the condition is true

Continue - Stops the current iteration and continues with next

Pass - When loops are empty, it throws an error. To avoid errors, pass statement is used

Functions: It's a block of code which gets executed only when the function name is called.

Parameters are passed as inputs.

Advanced function:

1. *args and *kwargs (non-keyword arguments and keyword arguments respectively)

- These functions are used when we are not sure about the number of input parameters to pass in the function.
- *args passes arguments as tuple and *kwargs passes arguments as dictionary

2. Yield Function: Yield return generator object. It is used to iterate over a sequence.

3. Return function: Return function ends the function execution and returns results.

4. Generators: When a function iterates over a sequence, the generator returns a sequence of values.

Function Keywords:

1. Lambda: Its an anonymous function which takes multiple arguments and performs one operation

Syntax → lambda : expression

2. Global and local variables:

- Global variables are defined outside the function and can be accessed throughout the program and also inside the function
- Local variables are defined inside a function and can be accessible only within it.

Functions Memoization:

1. Decorators: They are used when any modification is needed in the behavior of the function without changing the function.

2. Memoize: it is the method to store results of previous function calls in order to speed up the future calculations.

String: It is a collection of numbers, alphabets or other characters within quotes.

1. Slicing → Creating sub- strings from the given string

str[start:stop] → start to stop-1

str[start:] → start to end of string

str[:stop] → start to stop-1

str[:] → all string elements

str[start:stop:step] → start to stop by step

2. Modifying: Changing the string elements.

UpperCase, LowerCase, Removing space, Replacing characters

3. Concatenate: Merging 2 strings together using +, join

s1 = 'hello', s2 = 'world'

Result = 'helloworld'

Methods:

Result = s1+ s2

Result = "".join([s1,s2])

Result = '%s %s', % (s1, s2)

Result = f'{s1} {s2}'

Escape characters are used to insert illegal characters into a string which otherwise leads to an error.

\' - Single quote

\\ - Backslash

\n - Newline

\s - Space

\t - Tab

\b - backspace

String Methods: Built-in methods that can be used on strings

len(str) - length of string

str.lower() - converts all the character of the string to lowercase

str.upper() - converts all the character of the string to uppercase

min(str) - minimum alphabetical character in a string

max(str) - maximum alphabetical character in a string

List: Stores multiple items in a single variable of different types. It is mutable and ordered. []

- Accessing elements:

```
List = [1,2,3,4]
```

```
List[0] = 1
```

- Updating List:

```
List = [2,4,6,8]
```

```
List[2] = 3
```

```
Updated_list = [2,4,3,8]
```

- Remove list item:

```
List.remove(index_value)
```

- Looping lists:

```
List = [1,4,6,8]
```

```
For i in list:
```

```
    print(i)
```

```
L2 = [1,2,3,4,5]
```

```
j = 0
```

```
While j < len(L2):
```

```
    print(j)
```

```
    j = j+1
```

- List comprehension: shortest syntax of for loop. This can be used to create a list from other data types

```
List_1 = ['a','b','c']
```

```
[print(s) for s in List_1]
```

- Sorting List: Arranging list in ascending or descending order.

```
list.sort(reverse = True/False) → reverse = True means descending order
```

- Copy lists: To make a copy of the list

```
New_list = list1.copy()
```

```
New_list = list(list1)
```

- Join lists:

L1 = [1,2,3]

L2 = ['a','b']

- '+'

L3 = L1+L2 → [1,2,3,'a','b']

- append

For i in L2:

L1.append(i)

- extend:

For i in L2:

L1.extend(L2)

- Clear elements from the list

list.clear()

- To count specific element of a list

list.count(element)

Tuple: Stores multiple items in a single variable. It is immutable. ()

1. Accessing tuple elements - Similar to list

tup[start:stop] → start to stop-1

tup[start:] → start to end of string

tup[:stop] → start to stop-1

tup[:] → all string elements

tup[start:stop:step] → start to stop by step

2. Updating tuple:

Convert the tuple to list and then update the using index value. Convert the list back to tuple

3. Adding tuple to tuple → tup1 + tup2

4. Looping:

T1 = (1,2,3,4)

For i in T1:

print(i)

T2 = [1,2,3,4,5]

j = 0

While j < len(T2):

```
print(j)
j = j+1
```

Sets: Stores multiple items in a single variable. It is unordered. Items can be removed and added. {}

1. Adding elements to set:

```
set.add(new_element)
set.update(new_element)
```

2. Remove elements from set:

```
set.remove(set_element)
set.discard(set_element)
```

3. Pop - removes random elements from the set

```
set.pop()
```

4. To remove all the elements from the set

```
set.clear()
```

5. Looping

```
S = {1,2,3,4,5}
For i in S:
    print(i)
```

6. Updating a set: Combines 2 set elements excluding duplicates

```
S1 = {1,2,3,4}
S2 = {6,7,8,9}
S1.update(S2) → {1,2,3,4,6,7,8,9}
```

7. Union of set → Combines multiple set elements excluding duplicates

```
S1.union(S2,S3, S4)
```

8. To make a copy of the set → new_set = set.copy()

9. difference() → Returns the elements that are present only in set1 and not in set2

```
Set1 = {'a','b','c'}
Set2 = {'d','e','c'}
Res = Set1.difference(Set2) → {'a','b'}
```

Dictionary: Stores values as key-value pairs. It is ordered and mutable. {key: value}

1. Accessing:

```
D = {'a':1,'b':2,'c':3}
```

```
D['a'] = 1
```

```
D.get('a') = 1
```

```
D.keys() → returns all the keys → a,b,c
```

```
D.values() → returns all the values → 1,2,3
```

```
D.items() → returns key, value pairs as tuple in a list → [('a',1),('b',2),('c',3)]
```

2. Update:

```
D.update({key:value})
```

3. Adding items:

```
D['d'] = 23
```

```
D → {'a':1,'b':2,'c':3,'d':23}
```

4. Remove item:

```
D.pop('a') → {'b':2,'c':3,'d':23} → removes specific item based on key name
```

```
D.popitem() → {'b':2,'c':3} → removes the last item from the dictionary
```

```
del D['c'] → {'b':2}
```

```
D.clear() → {} → returns an empty dictionary
```

5. Looping:

```
1. D = {'a':1,'b':2,'c':3,'d':23}
```

```
For i in D:
```

```
    print(i) → returns all the keys
```

```
2. For i in D:
```

```
    print(D[i]) → returns all the values
```

```
3. For i in D.values():
```

```
    print(i) → returns all the values
```

```
4. For i, j in D.items():
```

```
    print(i,j) → return keys and values
```

6. Copy:

1. Shallow Copy: A copy of the original object is stored and only the reference address is finally copied.
2. Deep Copy: Copy of the original object and the repetitive copies both are stored.

7. Nested Dictionary: Dictionaries within another dictionary.

```
Dictionary = {"fruit1" : {"name" : "apple", "color" : "green"},  
              "fruit2" : {"name" : "strawberry", "color" : "pink"},  
              "fruit3" : {"name" : "orange", "color" : "orange"}}
```

Accessing items from nested dictionary: `print(dictionary["fruit1"]["name"])` → apple