Programming with Python

Friday, 8 August, 2025 11:13 PM

Basics :

- Code or Source Code - The sequence of instructions in a program.
- Syntax - The set of legal structures or commands that can be used in a particular programming language.
- Output - The messages printed to the user by a program.
- Console - The text box onto which the output is printed.

History of Python :

- First implemented in 1989 by Guido Van Rossum.
- Free, open-source software with community based development.
- Named after BBC show "Monty Python's Flying Circus".

Compiling and Interpreting :

- Many languages require you to compile (translate) your program into a form that the machine understands.
- Python is instead directly interpreted into machine language.

Indentation :

- Most programming languages use curly brackets {} to define a code or a block.
- Python uses indentation.
- The code of a block starts with an indentation and ends with the first un-indented line.
- Incorrect indentation will result in Indentation Error.

Interactive - You directly type to python 1 line at a time and it responds.

Script - You enter a sequence of statements (lines) into a file using a text editor and you tell python to execute the statements in t he file.

Variable :

- It is a name that's used to refer to the memory location.
- Python variable is also called as an identifier which is used to hold value.
- No need to specify its type while defining the variable as python gets the variable type when we assign value to it,
  - String - ""
  - Integer - Whole Numbers
  - Float - Number with decimal values
- Variable names can be a group of both letters and numbers <u>but must start with a letter or an underscore</u>.
  - It is <u>case sensitive</u> (Name and name are 2 different variables).
  - The name <u>should not</u> contain any special characters or whitespaces.
  - The name <u>should not</u> be similar to any keywords.

- Global -
  - They can be used throughout the entire program (Can be used inside and outside a function).
  - Variable declared outside a function is considered as global variable by default.
    - We have to use the **global** keyword for the variable declared inside a function to make it accessible outside the function.
  - Eg.

```
a = 50
def add():
    b=10
    global c
    c=5
    d=a+b+c
    print(d)
add()
print(a, c)
```

- Local -
  - Variables that's defined inside a function are known as local variables.
  - Eg.

```
def add():
    a=10
    b=20
    c=a+b
    print(c)
add()
```

Constants :

It is a type of variable who's value cannot be changed (containers that hold information that cannot be changed later).
They are usually declared and assigned in a module.
Declaring and assigning -

Create python file "constant.py",
```
PI = 3.14
```
In another python file import the constant,
```
import constant
print(constant.PI)
```

Data Types :

Variables can hold values and every value got a data type.
User uses type() function to find the type of the value stored by the variable (User cannot define any data type).
Basic types -

Type	**Declaration**	**Example**	**Usage**
String	str	"Hello World"	Used for letters
Integer	int	1234	Used for whole numbers
Float	float	12.34	Used for numbers with decimal values
Boolean	bool	True or False	Used for conditional statements
NoneType	None	None	Used for empty variables

Important for arithmetic operations.

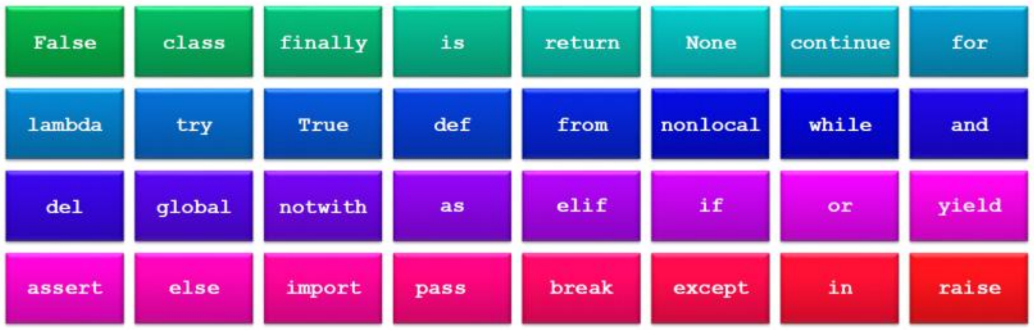
They can be converted from str to int or to float in any order.

Have to use str(), int(), float().

Eg.
```
x=int(input("Enter a number"))
print(type(x))                        #Type => Integer
y=str(x)
print(type(y))                        #Type => String
```

Keywords :

They are special reserved words that convey a special meaning to the compiler/interpreter.
Each keyword has a special meaning or operation.
They cannot be used as variables.



False	class	finally	is	return	None	continue	for
lambda	try	True	def	from	nonlocal	while	and
del	global	notwith	as	elif	if	or	yield
assert	else	import	pass	break	except	in	raise

Arithmetic/Numeric Operations :

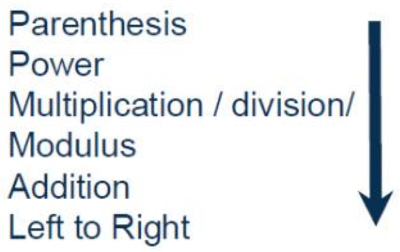
Symbol	**Task performed**
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Floor Division
%	Modulus
**	Power of

Modulus gives the reminder after division.

Precedence Order Rules :

Highest Precedence rule to Lowest Precedence rule -

Parenthesis are always respected.
Exponentiation (Raised to a power).
Multiplication, Division, Reminder.
Addition, Subtraction.
Left to Right.



(It goes from top to bottom)

Control Structures :

Conditional Statements -

Structure -

One-way decision - IF

Eg.
```
height=int(input("Enter Your Height : "))
if height<=140:
        print("You can enjoy a free ride")
        price=0
```

Two-way decision - IF, ELSE

Eg.
```
num=int(input("Enter a number : "))
if num<0:
        print("Number is negative.")
else:
        print("Number is positive.")
```

Multi-way decision - IF, ELIF, ELSE (Else is optional as Elif works as Else and If)

Eg.
```
if x>8:
        print("Good")
elif x>4:
        print("Passed")
else:
        print("Failed")
```

Nested decision

Eg.
```
if x<=15:
        if x==10:
                print("Number is 10")
        else:
                print("Number is lesser than 15")
else:
        print("Number is more than 15")
```

Repetition Structures :

While loop (Indefinite) - Can be used when we do not know how many times the loop exists (condition is set which stops the loop).

Eg.
```
i=1
while i<=3:
        print("i = ", i)
        i=i+1                              # If we miss this, then it becomes an infinite loop
print("Done")
```

For loop (Definite) - Used when we know how many loops to run (Number of loops is known).

Eg i.
```
list=eval(input("Enter a list"))
for i in list:
        print(i)
```

```
Eg ii.  for I in range(1, 5):
            print(i)
```

Range -

Method	Description
for i in range(5):	Range starts at 0 and ends at 4.
for i in range(1, 5):	Range starts at 1 and ends at 4.
for i in range(1, 5, 1):	Range starts at 1 and ends at 4, and it takes values by skipping 1 value
for I in range(5, 1, -1)	Range starts at 5 and ends at 2, and it goes in reverse => 5, 4, 3, 2.

range(1, 5, -1) -> will give error cause -1 is reverse and starting from 1 and in reverse is 0 which will be out of range.

Sentinel Loops - A loop that continues to process the data until it reaches a special value which signals the end.
The special value is called as *sentinel*.
It must be disguisable from the data as it is not processed as part of the data.

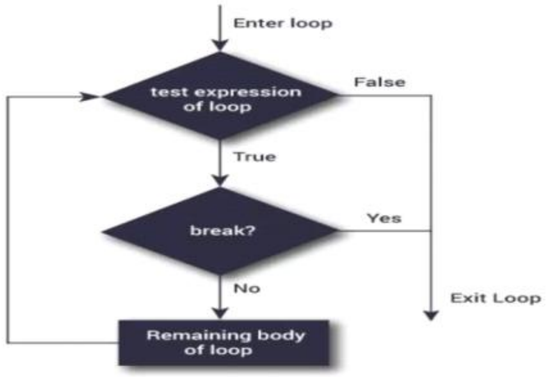
```
Eg.   count=0
      while count<5:
            print(count, "Is lesser than 5")
            count=count+1
      else:
            print(count, "is not lesser than 5")
```

Nested Loops - One loop executes inside another loop.

```
Eg i.  i = 1
      while (i <= 2):
            j = 1
            while (j <= 3):
                  print("i = ", i, " j = ", j)
                  j = j + 1
            i = i + 1
      print("Done!")
```

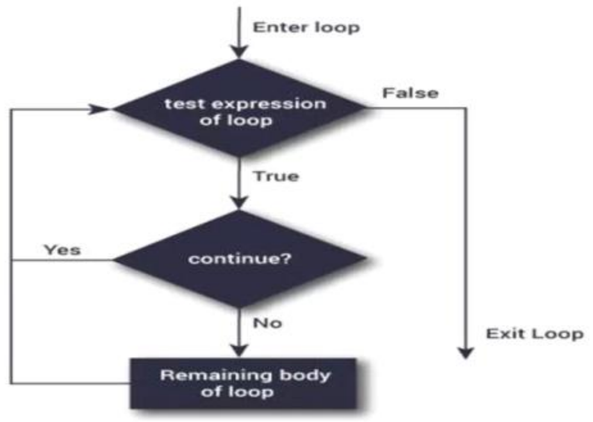
Break Statements - Terminates the loop containing it.

```
Eg.   for val in "string":
            if val == "i":
                  break
            print(val)
      print("The end")
```



Continue Statement - Used to skip the rest of the code inside a loop for current iteration only.
Loop doesn't terminate but it continues with next iteration.

```
Eg.   for val in "string":
            if val == "i":
                  continue
            print(val)
      print("The end")
```



Infinite loop - Runs forever (have to be careful while doing loops so that this doesn't occur)
A loop that doesn't have a way to stop the loop.

Pass Statement - For an empty block of code use **pass** statement (It is **null** statement).
Difference between pass and a comment is that comment is ignored by python but pass statement isn't ignored.
Nothing happens when it is executed.
Eg.
```
if height>140:
        pass
print("You have to pay the ticket price")
```

Functions :
Breaking programs into manageable small chunks of code.
Can be called when we want that particular code.
Reduces complexity of the code.
Easier to maintain and understand.
There is 2 types of functions,
Built-in, (few important ones)

Method	**Description**
type(x)	Gives the type of the variable. Like str, int, float
str(x)	Converts the variable into a string.
int(x)	Converts the variable into a integer. (Can convert only float or string with numbers, Strings with words cannot be converted into an integer).
float(x)	Converts the variable into a float. (Can convert only integer or string with numbers, Strings with words cannot be converted into an integer).
x.upper()	Converts the string into uppercase. Eg : hello world => HELLO WORLD
x.lower()	Converts the string into lowercase. Eg : HELLO WORLD => hello world
x.title()	Converts the string into title. Eg : hello world => Hello World
len()	Gives the length of the string.

Note : x is the variable name

For math function we have to *import math.*

User Defined,
We must define the function before using them in the main body.
Later in the main body we call the function using its name.
Eg:

```
def Add():                        #Defining the function Add()
    a=10
    b=5
    c=a+b
    print(c)

Add()                             #Calling the function Add() in the main code
```

Parameters are variables that we use in a function definition.
Argument is the value that we pass into the function as its input when the function is called.

Passing Arguments into a function,

Pass by Reference,
The actual object is passed into the function.
Object is mutable type, so all changes made to the object inside the function affects the value.
Eg:

```
my_list=[1,2,4,5,6]
def passRef(my_list):
    my_list[3] = 30
    print("Value inside the function, my_list = ", my_list)
print("before the function call, my_list= ", my_list)
passRef(my_list)
print("after the function call, my_list= ", my_list)
```

Pass by Value,
Function creates a copy of the variable passed to it as an argument.
Object is immutable type, so all changes made inside the function doesn’t affect the actual value.
Eg:

```
a=10
def passValue(a):
  a=20
  print("inside function, a= ", a)
  return
print("before function call, a= ", a)
passValue(a)
print("after function call, a= ", a)
```

function(*vari), In this the * means the function can take in 0 or more arguments.

Fruitful Function - A function that produces an end result (or returns a value).
Void (Non-Fruitful) Function - A function that doesn’t return a value.

No function overloading (each function has a unique name). If a function name is repeated, the function is overwritten.

Strings :
Strings are immutable.
Strings Concatenation,
Joining 2 or more strings together is called concatenation.
+ Operator is used for joining 2 or more strings together.
Eg :

```
a="Hello"
b= a + "There"
print(b)
c= a + " " + "There"
print(c)
```

* Operator is used to repeat the string the given number of times.
Eg :

```
a="Hello"
print(a * 3)
```

Reading a string,

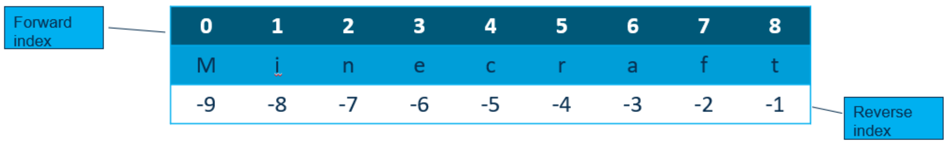
Eg:

```
name = input("Enter your name :")
print(name)
```

Converting,

Method	**Description**
str(x)	Converts the variable into a string.
int(x)	Converts the variable into a integer. (Can convert only float or string with numbers, Strings with words cannot be converted into an integer).
float(x)	Converts the variable into a float. (Can convert only integer or string with numbers, Strings with words cannot be converted into an integer).

Each character in a string has an index.



0	1	2	3	4	5	6	7	8
M	i	n	e	c	r	a	f	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Forward Index starts at 0
Reverse Index starts at -1

To access each character, you call them using the index number.

Eg :

```
a="Hello"
print(a[0])
print(a[1])
print(a[2])
print(a[3])
print(a[4])
```

In which 'a' is the variable name and the numbers inside the [ ] specify the index number.

Special Characters,

The backslash (\ ) is used to introduce a special character.

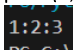
Escape Sequence	Meaning
\\	Backslash
\'	Single Quote
\"	Double Quote
\n	New Line
\t	Tab

Print Separator and End,

By default print separates the values with a space.
This can be changed into anything that we want.

Eg :

```
print(1, 2, 3, sep=":")
```

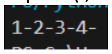


```
1:2:3
```

When having looped print statements, they print on the next line by default, which can be changed.

Eg :

```
for i in range(1, 5):
   print(i, end="-")
```



```
1-2-3-4-
```

in Keyword can be used as an operator,

The **in** Keyword can be used to check if a particular string is in a string.

Eg :

```
a="Hello"
print("e" in a)
```

len() function gives the length of a particular string.

Eg :

```
a="Hello"
print(len(a))
```

Iterating over strings,

Eg :

```
a="Hello"
for i in a:
   print(i)
```

Slicing a String.

Slicing means cutting a string into 2 or more.

Eg :

```
x="Hello"
newx=x[:3]
```

Method	**Description**
x[:3]	Starts at 0 and ends at 3rd position.
x[3:]	Starts at 3 and ends at the last index.
x[:-1]	All characters except the last.
x[::-1]	Reverses the string.

Strings are Objects (Immutable),

Method	**Description**
x.upper()	Converts the string into uppercase. Eg : hello world => HELLO WORLD
x.lower()	Converts the string into lowercase. Eg : HELLO WORLD => hello world
x.title()	Converts the string into title. Eg : hello world => Hello World
x.capitalize()	Capitalizes the first character of the string. Eg : hello world => Hello world
x.count(a)	Counts the number of given argument 'a' in the string. ('a' is an argument that we have to enter).
x.islower()	Checks if the string is in lowercase or no. (Passes Boolean expression).
x.isupper()	Checks if the string is in uppercase or no. (Passes Boolean expression).
x.swapcase()	Swaps the case for the string. Eg : Hello World => hELLO wORLD
x.replace(a, b)	Replaces the character 'a' with character 'b' in the string. ('a' and 'b' are arguments that we have to enter).
x.find(a)	Finds and prints the position of the character 'a' in the string. ('a' is an argument that we have to enter).
x.lstrip()	Strips the whitespace on the left side. Eg : " Hello World " => "Hello World "
x.rstrip()	Strips the whitespace on the right side. Eg : " Hello World " => " Hello World"
x.strip()	Strips the whitespace on both the sides. Eg : " Hello World " => "Hello World"
max(x)	Print the max alphanumeric characters in the string.
min(x)	Print the min alphanumeric characters in the string.

Note : x is the variable name

Lists [ ] :

Lists are mutable.

A kind of a Collection, in which a single variable can hold many values (Data inside [ ] brackets).

Eg : x=[3, "hello", True]

One list can contain another list (sub list).

Eg : x = [3, 4, [5, 6], 7]

A list can store any type of data.

Creating a list.

Can be entered directly.

Entered using list() function.

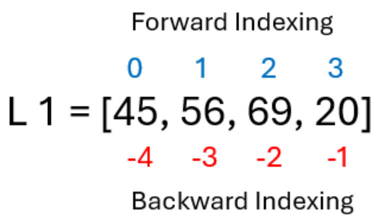
Accessing elements in a list.

Using an index operator to access an element in the list.

Eg :

```
x=[1, 2, 3, 4, 5, 6, 7, 8 , 9]
print(x[1])                    #Where x[1] specifies the index value to be called in the list x
```

If the index value is negative, counting starts in the reverse

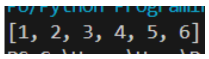


Using For loops,

Eg:

```
x=[1, 2, 3, 4, 5]
for i in x:
    print(i)
```

Adding elements into a list.

Method	Description
x.append(item)	Added into the end of the list. Where 'a' is a value.
x[4]=123	Value at the 4th index (5th element) of the list is modified.
x.extend(a)	Adds another list's values directly into the list. Where 'a' is another list.
Concatenation using +	You can add to lists together using + operator. Eg : a=[1, 2, 3] b=[4, 5, 6] c=a+b print©  [1, 2, 3, 4, 5, 6]

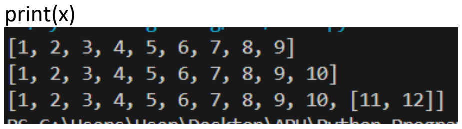
Difference between append and extend,

Append adds the data passed directly into the list.

If a list is passed, it adds it as a sub list.

Eg:

```
x=[1, 2, 3, 4, 5, 6, 7, 8 , 9]
print(x)
x.append(10)
print(x)
x.append([11, 12])
print(x)
```

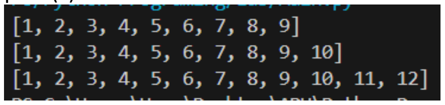


```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, [11, 12]]
```

While Extend adds the data in the list directly (extend only works if the passed data is another list).

Eg:

```
x=[1, 2, 3, 4, 5, 6, 7, 8 , 9]
print(x)
a=[10]
x.extend(a)
print(x)
b=[11, 12]
x.extend(b)
print(x)
```



```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

If we kept "a=10" instead of "a=[10]", it raises and error.

Difference between Append and Concatenate.

Concatenate adds 2 different lists into 1, using + operand. Concatenate adds only if u pass 2 lists and it raises an error if u pass integer or string.

Eg:

Eg :

```
a=[1, 2, 3]
c = a + [4]
print(c)
```

If we write "c = a + 4" it will raise an error.

Slicing lists.

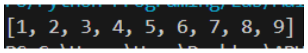
Slicing means cutting a List into 2 or more (same as strings).

Eg :

```
x=[1, 2, 3, 4, 5]
newx=x[:3]
```

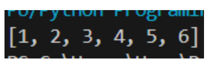
Method	**Description**
x[:3]	Starts at 0 and ends at 3rd position.
x[3:]	Starts at 3 and ends at the last index.
x[:-1]	All characters except the last.
x[::-1]	Reverses the list.

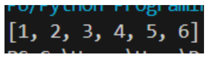
Methods,

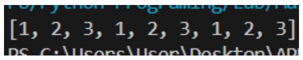
Method	**Description**
list.append(item)	Appends the item to the end of the list.
list.insert(index, item)	Inserts the item in the specific index.
list.remove(item)	Removes the first occurrence of the item from the list.
list.pop(index)	Removes the data at that index.
list.extend(a)	Appends a list into the list. Where 'a' is another list.
list.count(element)	Shows how many times the element is repeated in the list.
list.sort()	Sorts the list into ascending order. Eg : x=[9, 2, 5, 7, 6, 1, 3, 8, 4] x.sort() print(x)  [1, 2, 3, 4, 5, 6, 7, 8, 9]
list.reverse()	Reverses the list. (Can do y=x[::-1] also).
len(list)	Prints the length of the list.
List.index(a)	Shows the index value of the element in the list.

Mathematical Operations,

Symbol	Description

+	Returns a new list by adding 2 lists. (Appends 2 lists together).
	Eg :
	a=[1, 2, 3]
	b=[4, 5, 6]
	c=a + b
	print(c)
	
	`[1, 2, 3, 4, 5, 6]`

+=	Adds list2 into list1. (Works like extend()).
	Eg :
	a=[1, 2, 3]
	b=[4, 5, 6]
	a+=b
	print(a)
	
	`[1, 2, 3, 4, 5, 6]`

*	Multiplies a list with an integer 'n'.
	Eg:
	a=[1, 2, 3]
	b=a*3
	print(b)
	
	`[1, 2, 3, 1, 2, 3, 1, 2, 3]`

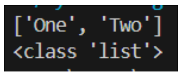
Ranges are Lists.

range(5)	=>	[0, 1, 2, 3, 4]
range(2, 7)	=>	[2, 3, 4, 5, 6]
range(0, 10, 2)	=>	[0, 2, 4, 6, 8]

String.split() creates a List.

Eg :

```
string="One Two"
list=string.split()
print(list)
print(type(list))
```



```
['One', 'Two']
<class 'list'>
```

Tuple ( ) :

Tuples are Immutable.

So you cannot alter its contents like in lists.
x.sort() , x.reverse() , x.append(a) , x.extend(a) and all raises errors.
Where 'x' is a tuple and 'a' is an element.

If the elements do not change, then use tuples to avoid data being changed accidently.

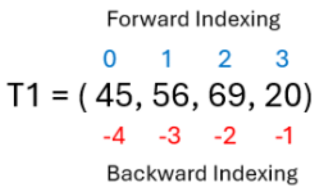
Accessing a Tuple,

You use indexing to access a tuple's value (similar like lists and strings).

Eg:

```
x=(1, 2, 3, 4, 5, 6, 7, 8 , 9)
print(x[1])                    #Where x[1] specifies the index value to be called in the tuple x
```

If the index value is negative, counting starts in the reverse



Using For loops,

Eg:

```
x=(1, 2, 3, 4, 5)
for i in x:
    print(i)
```

Slicing a Tuple,

Cutting a Tuple into 2 or more (same as lists and strings).

Eg :

```
x=(1, 2, 3, 4, 5)
newx=x[:3]
```

Method	Description
x[:3]	Starts at 0 and ends at 3rd position.
x[3:]	Starts at 3 and ends at the last index.
x[:-1]	All characters except the last.
x[::-1]	Reverses the tuple.

Methods,

Method	Description
len(x)	Prints the length of the tuple.
x.count(a)	Shows how many times the element is repeated in the list.
x.index(a)	Shows the index value of the element in the tuple.

Note : 'x' is the tuple name and 'a' is the element.

There is no x.sort() , x.reverse() , x.append(a) , x.extend(a) since tuples are immutable.

Tuples are comparable.

The comparison operators work with tuples.

Operator	Action
<	Lesser Than
>	Greater Than
==	Equal To
<=	Lesser Than or Equal To
>=	Greater Than or Equal To
!=	Not Equal

Eg. i :

```
x=(7, 1, 2) < (5, 0, 2)
print(x)
```

Output is False, as python checks only the first values (7<5) since its False it prints False and doesn’t check any further.

```
x=(0, 1, 2) < (5, 0, 2)
print(x)
```

Output is True, as python checks only the first values (0<5) since its true it prints True and doesn’t check any further.

Eg. ii.

```
x=(5, 1, 2) == (5, 0, 2)
print(x)
```

Output is False, as python first checks for the first values (5==5) since its True it checks for the next value (1=0) since its False, the whole value becomes false.

Note : For all comparison, final value is True only if all the values are True. Even if 1 value is False the whole value is False.

Tuples are more efficient than lists.

Since tuples are immutable, they only got few methods which results in small object size as python knows the data won't change and python accesses tuples faster as they are optimized for small storage space)

Meanwhile lists are mutable, so they have a lot of methods to alter the data and consumes higher memory.

Set { }:

Sets are Mutable.

Important Feature of a set is that the <u>data never repeats</u>.

Creating a Set.

All the elements are placed inside the curly brackets { } separated by comma (,).

Or they are created using built-in function set().

Elements,

A set can contain elements of different types (string, integer, float, tuple).

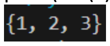
Eg :

```
x={1, 2.5, "Hello", (5, 8)}
print(x)
```

A set cannot contain a list but a list can be converted into a set using the set() function.

Eg:

```
x=[1, 2, 3]
print(set(x))
```



```
{1, 2, 3}
```

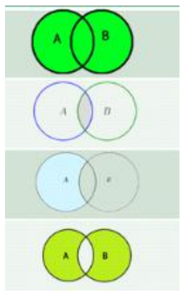
Methods,

Method	Description
len(x)	Prints the length of the list.
x.add(a)	Adds the element into the set at the end. Used when adding only 1 element.
x.update(a)	Adds the element into the set. Used when adding more than 1 element
x.remove(a)	Removes the element from the set. It raises an error if the element is not present in the list.
x.discard(a)	Removes the element from the set. It doesn't raise an error if the element is not present in the list.
y=x.copy()	Makes a copy of the set.
x.clear()	Clears the data of the set.

Note : 'x' is the set name and 'a' is the element.

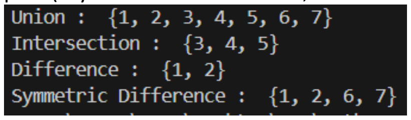
Operations,

Method	Operator	Description
Union	|	Contains all elements that are in set A or in set B.
Intersection	&	Contains elements that are in both set A and set B.
Difference	-	Contains all elements that are in set A but not in set B.
Symmetric_Difference	^	Contains all elements that are either in, In set A but not in set B In set B but not in set A



Eg :

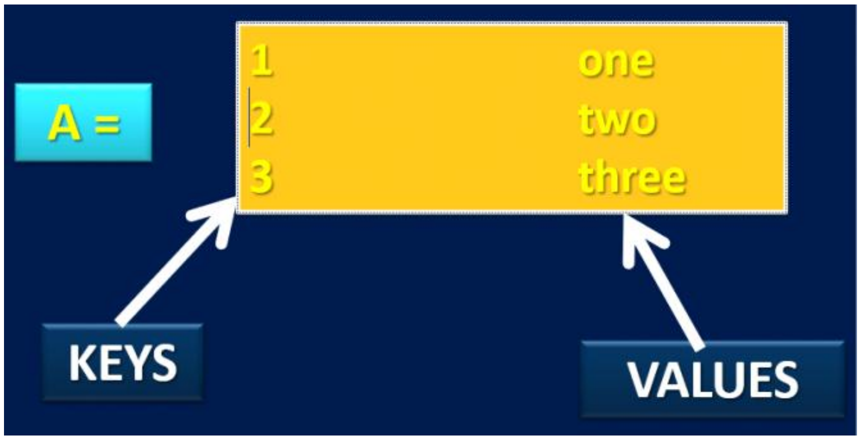
```
setA = {1,2,3,4,5}
setB = {3,4,5,6,7}
print("Union : ", setA|setB)
print("Intersection : ", setA & setB)
print("Difference : ", setA - setB)
print("Symmetric Difference : ", setA ^ setB)
```



```
Union :  {1, 2, 3, 4, 5, 6, 7}
Intersection :  {3, 4, 5}
Difference :  {1, 2}
Symmetric Difference :  {1, 2, 6, 7}
```

Dictionaries { } :

Dictionaries are like a database in python.
They are Mutable.
They store values as a pair of entries.
Each Pair of entry contains,
A Key.
A Value.
Pairs of entries are separated by commas. While Key and Value are separated by colon (:) .

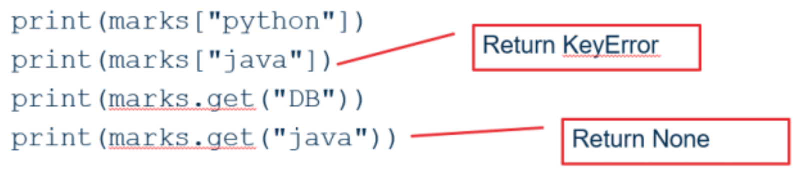


Accessing a Dictionary,

Keys are used to access the values in a dictionary instead of indexes like others.

Keys are used inside [ ] brackets or used inside get() function.

The get() function returns None instead of Error if the key is not found.



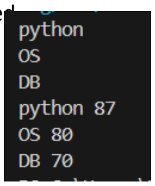
```
print(marks["python"])
print(marks["java"])
print(marks.get("DB"))
print(marks.get("java"))
```

Return KeyError

Return None

Adding Elements into a dictionary,

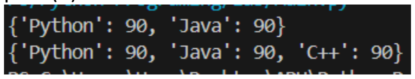
If the key already exists, the value gets updated. Or a new key : value pair is added.



```
python
OS
DB
python 87
OS 80
DB 70
```

Eg :

```
d={"Python":80, "Java":90}
d["Python"]=90
print(d)
d["C++"]=90
print(d)
```



```
{'Python': 90, 'Java': 90}
{'Python': 90, 'Java': 90, 'C++': 90}
```

Deleting Elements from a dictionary,

A particular item can be deleted using the pop() function.

Eg:

```
d={"Python":80, "Java":90}
d.pop("Python")
print(d)
```

All the items can be removed at once using the clear() function.

Can use del keyword to remove individual items or the entire dictionary itself.

popitem() can be used to remove the last inserted value. It raises Error if the dictionary is empty.

Iterating through the dictionaries,

Using For Loop,

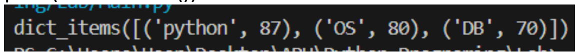
Eg:

```
marks = {"python": 87, "OS": 80, "DB": 70}
for subject in marks:                    #Prints only the key
   print(subject)
for subject in marks:                    #Prints the key and the value
   score = marks[subject]
   print(subject, score)
```

Use items() function.

Eg:

```
marks = {"python": 87, "OS": 80, "DB": 70}
print(marks.items())
```



```
dict_items([('python', 87), ('OS', 80), ('DB', 70)])
```

Checking Frequency of elements.

We can check how many of the characters are present in the particular string.

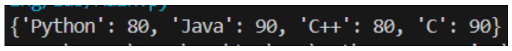
Eg :

```
text = "Python Dictionary is Awesome"
freq = {}
for ch in text:
   freq[ch] = freq.get(ch, 0) + 1
for key in freq:
   print(key, freq[key])
```

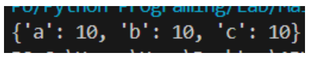
In this, Capital and Lowercase characters are counted separately (Spaces are also counted).

Methods,

Method	**Description**
x.clear()	Removes all the values from the dictionary.
x.copy()	Copies the dictionary into another variable.
x.get(key[a])	Returns the value of the key. Returns None if the key doesn't exist.
x.items()	Returns a new view of the dictionary's items (key, value).
x.keys()	Returns a new view of the dictionary's keys.
x.values()	Returns a new view of the dictionary's values.

x.update()	Updates the dictionary with key/value pair from other, existing keys will be overwritten.
	Eg:
	x={"Python":80, "Java":90}
	x.update([("C++", 80), ("C", 90)])
	print(x)
	
	`{'Python': 80, 'Java': 90, 'C++': 80, 'C': 90}`

x.pop(key)	Removes the item from the dictionary.
x.popitem()	Removes the last inserted value.

x=dict.fromkeys([seq], v)	Creates a dictionary with uniform pre-defined values.
	Eg:
	x=dict.fromkeys(["a", "b", "c"], 10)
	print(x)
	
	`{'a': 10, 'b': 10, 'c': 10}`

len(x)	Prints the length of the dictionary.
sorted(x)	Returns a new sorted list of keys in the dictionary.

Difference between List, Tuple, Dictionary.

List	**Tuple**	**Dictionary**
Uses [ ] Brackets.	Uses ( ) Brackets.	Uses { } brackets.
Mutable. Data can be edited.	Immutable. Data cannot be edited.	Mutable. Data (only Values) can be edited.
		Every entry has a Key and a Value (pair).
Can hold any type of value. Strings, Integers, Float, Lists (sub), Tuples, Dictionaries, Sets.	Can hold any type of value. Strings, Integers, Float, Lists, Tuples (sub), Dictionaries, Sets.	Can hold any type of value. Strings, Integers, Float, Lists, Tuples, Dictionaries (sub), Sets.
Elements are accessed using indexing.	Elements are accessed using indexing.	Elements are accessed using keys.
Used to store and alter values.	Used to store pre-defined values that won't change.	Used as a database.