Introduction to Python

Python Basics:

Python Data Types:

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data.

Following are the built-in data types in Python:

Number → Integer, Complex Numbers, Float

Sequence Type → String, List, Tuple

Boolean

Set

Dictionary

Binary Types → Bytearray, Bytes, Memoryview

Number Type :

There are two main number type i.e. Integer and Float.

(Complex Number is also considered in the number data type)

```
DataTypes.py > ...
x = 5
y = 10.5

print(type(x))
print(type(y))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
<class 'int'>
<class 'float'>
shtlp_0061@SHTLP0061:~/Python Basics$
```

Sequence Type:

1) Strings →

Strings are ordered sequences of character, using the syntax of either single quotes or double quotes.

‘Hello’ or “Hello”

```
DataTypes.py > ...

#2) String data type

s = "hello"
print(s)
print(type(s))
print(len(s))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
hello
<class 'str'>
5
shtlp_0061@SHTLP0061:~/Python Basics$
```

Indexing → allows you grab a single character from string

ex → (h, e, l, l, o → 0, 1, 2, 3, 4) respective indexes

(h, e, l, l, o → 0, -4, -3, -2, -1) reverse indexing

DataTypes.py > ...

```python

#2) String data type

s = "hello"
print(s)
# print(type(s))
# print(len(s))

print(s[1])
print(s[0])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
hello
<class 'str'>
5
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
hello
e
h
shtlp_0061@SHTLP0061:~/Python Basics$
```

Slicing → allows you to grab a sub-section of multiple characters

syntax : **[start:stop:step]**

start : start index for slicing

stop: index you will go up-to (not included)

step: size of jump you take

(to reverse string → s[::-1])

```
DataTypes.py > ...

print(s[1:])
# it will start printing from index 1 to end

print(s[:3])
#it will start priting from start to index 2 (stop index is not included)

print(s[3:6])
print(s[::])
print(s[2:7:2])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
helloWorld
elloWorld
hel
loW
helloWorld
loo
shtlp_0061@SHTLP0061:~/Python Basics$
```

String concatenation →

```
DataTypes.py > ...

#string concatenation
a = "Hello"
b = "World"
c = a + b
print(c)

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
HelloWorld
Hello World
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
```

String format →

we cannot combine strings and numbers like this:

age = 22

txt = “my name is riya, I am ”+ age

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are.

DataTypes.py > ...

```python
# string format

age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))


quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))

# You can use index numbers {0} to be sure the arguments are placed in the correct pla
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
My name is John, and I am 36
I want 3 pieces of item 567 for 49.95 dollars.
I want to pay 49.95 dollars for 3 pieces of item 567.
shtlp_0061@SHTLP0061:~/Python Basics$
```

Escape Character →

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

```
DataTypes.py > ...
# escape character
txt = "We are the so-called \"Vikings\" from the north."
print(txt)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
We are the so-called "Vikings" from the north.
shtlp_0061@SHTLP0061:~/Python Basics$
```

2) List →

List are ordered sequences that can hold a variety of object types.

```
DataTypes.py > ...

#   list data type
mylist = [1,2,3]
print(mylist)
mylist = ["hello","world","bye"]
print(mylist)
mylist = [1,"hello"]
print(mylist)


# list slicing
# print(mylist[1:])
# print(mylist[0:2])
# print(mylist[::2])
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
[1, 2, 3]
['hello', 'world', 'bye']
[1, 'hello']
shtlp_0061@SHTLP0061:~/Python Basics$
```

List Slicing → same as string slicing

DataTypes.py > ...

```python
#   list data type
mylist = [1,2,3]
# print(mylist)
mylist = ["hello","world","bye"]
# print(mylist)
# mylist = [1,"hello"]
# print(mylist)


# list slicing
print(mylist[1:])
print(mylist[0:2])
print(mylist[::2])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
['world', 'bye']
['hello', 'world']
['hello', 'bye']
shtlp_0061@SHTLP0061:~/Python Basics$
```

List Concatenation →

```python
# list data type
mylist = [1,2,3]
# print(mylist)
mylist = ["hello","world","bye"]
# print(mylist)
# mylist = [1,"hello"]
# print(mylist)


# list slicing
# print(mylist[1:])
# print(mylist[0:2])
# print(mylist[::2])

# concatenation of list
list1 = ["my","name","is","riya"]

new_list = mylist + list1
print(new_list)
```

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
['hello', 'world', 'bye', 'my', 'name', 'is', 'riya']
shtlp_0061@SHTLP0061:~/Python Basics$
```

List Methods →

a) append() →

```python

# List Methods

# 1) append method
currencies = ['Dollar', 'Euro', 'Pound']
currencies.append('Yen')
print(currencies)

animals = ['cat', 'dog', 'rabbit']
wild_animals = ['tiger', 'fox']
animals.append(wild_animals)
print('Updated animals list: ', animals)

# 2)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
['Dollar', 'Euro', 'Pound', 'Yen']
Updated animals list:  ['cat', 'dog', 'rabbit', ['tiger', 'fox']]
shtlp_0061@SHTLP0061:~/Python Basics$
```

b) clear() →

```python

# 2) clear method
My_List = [1,2,3,4,5,6,7]
print(My_List)
My_List.clear()
print(My_List)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
[1, 2, 3, 4, 5, 6, 7]
[]
shtlp_0061@SHTLP0061:~/Python Basics$
```

c) sort() →

```
# Sort Method
prime_numbers = [11, 3, 7, 5, 2]
prime_numbers.sort()
print(prime_numbers)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
[2, 3, 5, 7, 11]
shtlp_0061@SHTLP0061:~/Python Basics$
```

d) copy() →

```
#copy method
prime_numbers = [2, 3, 5]
numbers = prime_numbers.copy()
print('Copied List:', numbers)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
Copied List: [2, 3, 5]
shtlp_0061@SHTLP0061:~/Python Basics$
```

e) insert() →

```python
# insert method
vowel = ['a', 'e', 'i', 'u']
# 'o' is inserted at index 3 (4th position)
vowel.insert(3, 'o')
print('List:', vowel)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
List: ['a', 'e', 'i', 'o', 'u']
shtlp_0061@SHTLP0061:~/Python Basics$
```

List Unpacking →

```python
# List Unpacking
colors = ['red', 'blue', 'green']
red = colors[0]
blue = colors[1]
green = colors[2]

print(red, blue, green)

r , b, g = colors
print(r,b,g)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
red blue green
red blue green
shtlp_0061@SHTLP0061:~/Python Basics$
```

None →

```python
# ****************************************************None*******************************************
x = None

if x:
  print("Do you think None is True?")
elif x is False:
  print ("Do you think None is False?")
else:
  print("None is not True, or False, None is just None...")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
None is not True, or False, None is just None...
shtlp_0061@SHTLP0061:~/Python Basics$
```

Dictionaries →

```python
# ********************************************Dictionary data type*************************************
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

print(type(thisdict))
print(thisdict)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
<class 'dict'>
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
shtlp_0061@SHTLP0061:~/Python Basics$
```

Note: Dictionaries cannot have two items with the same key.

Access items:

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

# print(type(thisdict))
# print(thisdict)

print(thisdict["brand"])

x = thisdict["model"]
print(x)
x = thisdict.get("model")
print(x)
x = thisdict.keys()
print(x)
y = thisdict.values()
print(y)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
Ford
Mustang
Mustang
dict_keys(['brand', 'model', 'year'])
dict_values(['Ford', 'Mustang', 1964])
shtlp_0061@SHTLP0061:~/Python Basics$
```

Update item:

```
DataTypes.py > ...
# print(y)

# update item
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
print(thisdict["year"])

thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})

print(thisdict["year"])
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
2018
2020
shtlp_0061@SHTLP0061:~/Python Basics$
```

Add item:

```
DataTypes.py > ...

# thisdict.update({"year": 2020})
# print(thisdict["year"])


# adding items
thisdict["color"] = "red"
print(thisdict)

thisdict.update({"color": "green"})
print(thisdict)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'green'}
shtlp_0061@SHTLP0061:~/Python Basics$
```

remove item →

pop() → The `pop()` method removes the item with the specified key name

popitem() → The `popitem()` method removes the last inserted item

del → The `del` keyword removes the item with the specified key name

clear() → The `clear()` method empties the dictionary

```
DataTypes.py > ...
# thisdict.update({ "color" : "green" })
# print(thisdict)


# remove item
thisdict.pop("model")
print(thisdict)

thisdict.popitem()
print(thisdict)

del thisdict["brand"]
print(thisdict)

thisdict.clear()
print(thisdict)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
{'brand': 'Ford', 'year': 1964, 'color': 'red'}
{'brand': 'Ford', 'year': 1964}
{'year': 1964}
{}
shtlp_0061@SHTLP0061:~/Python Basics$
```

Loop in dictionary →

```
DataTypes.py > ...
for x in thisdict:
  print(thisdict[x])
print('\n')

for x in thisdict.values():
  print(x)
print('\n')

for x in thisdict.keys():
  print(x)
print('\n')

for x, y in thisdict.items():
  print(x, y)
print('\n')
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
brand
model
year
color


Ford
Mustang
1964
red


Ford
Mustang
1964
red


brand
model
year
color


brand Ford
model Mustang
year 1964
color red


shtlp_0061@SHTLP0061:~/Python Basics$
```

Nested Dictionary →

```
DataTypes.py > ...
# nested dictionary
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

print(myfamily)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
shtlp_0061@SHTLP0061:~/Python Basics$
```

Dictionary methods →

1) clear()

2) copy()

3) items()

4) pop()

5) popitem()

Tuple →

A tuple is a collection which is ordered and **unchangeable**.

we cannot change, add or remove items after the tuple has been created.

```python
# ******************************************************Tuples data type*********************************************
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)

print(thistuple[1])
print(thistuple[-1])
print(thistuple[2:])

thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
('apple', 'banana', 'cherry', 'apple', 'cherry')
banana
cherry
('cherry', 'apple', 'cherry')
Yes, 'apple' is in the fruits tuple
shtlp_0061@SHTLP0061:~/Python Basics$
```

Update tuples →

Tuples are **unchangeable**, or **immutable** as it also is called.

Note: You can convert the tuple into a list, change the list, and convert the list back into a tuple.

x = ("apple", "banana", "cherry")

y = list(x)

y[1] = "kiwi"

x = tuple(y)

Unpack tuple →

```python
# unpack tuple
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
apple
banana
cherry
shtlp_0061@SHTLP0061:~/Python Basics$
```

Loop in tuples →

```python
# loop tuple
for x in thistuple:
  print(x)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
apple
banana
cherry
apple
cherry
shtlp_0061@SHTLP0061:~/Python Basics$
```

Join Tuple →

```python
# join tuple
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
('a', 'b', 'c', 1, 2, 3)
shtlp_0061@SHTLP0061:~/Python Basics$
```

Tuple methods →

count() →Returns the number of times a specified value occurs in a tuple

index() →Searches the tuple for a specified value and returns the position of where it was found

Sets →

A set is a collection which is *unordered*, *unchangeable**, and unindexed.

Note: Sets cannot have two items with the same value.

Note → The object in the `update()` method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

Remove method →

pop() →Remove a random item by using the `pop()` method

del → The `del` keyword will delete the set completely

clear() → The `clear()` method empties the set

Join methods →

The `intersection()` method will return a *new* set, that only contains the items that are present in both sets.

The `intersection_update()` method will keep only the items that are present in both sets.

stringIndexing.py | DataTypes.py

DataTypes.py > set1

```python
# ************************************************************sets data type*****************
thisset = {"apple", "banana", "cherry"}
print(thisset)
print(len(thisset))
print(type(thisset))

thisset = set(("apple", "banana", "cherry"))
print(thisset)

# access item
for x in thisset:
  print(x)

# add item
thisset.add("orange")
print(thisset)

tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)

# remove item
thisset.remove("banana")
print(thisset)

thisset.discard("banana")
print(thisset)

x = thisset.pop()
print(x)
print(thisset)

thisset.clear()
print(thisset)

thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)

# join sets
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

stringIndexing.py DataTypes.py

DataTypes.py > ...

```python

# remove item
thisset.remove("banana")
print(thisset)

thisset.discard("banana")
print(thisset)

x = thisset.pop()
print(x)
print(thisset)

thisset.clear()
print(thisset)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
{'banana', 'cherry', 'apple'}
3
<class 'set'>
{'banana', 'cherry', 'apple'}
banana
cherry
apple
{'banana', 'cherry', 'apple', 'orange'}
{'cherry', 'mango', 'apple', 'papaya', 'banana', 'orange', 'pineapple'}
{'cherry', 'mango', 'apple', 'papaya', 'orange', 'pineapple'}
{'cherry', 'mango', 'apple', 'papaya', 'orange', 'pineapple'}
cherry
{'mango', 'apple', 'papaya', 'orange', 'pineapple'}
set()
{1, 2, 3, 'b', 'c', 'a'}
shtlp_0061@SHTLP0061:~/Python Basics$
```

Boolean:

Boolean represent one of two values: True or False.

```
DataTypes.py > ...
65
66
67  # Boolean
68  print(10 > 9)
69  print(10 == 9)
70  print(10 < 9)
71
72  a = 200
73  b = 33
74
75  if b > a:
76    print("b is greater than a")
77  else:
78    print("b is not greater than a")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/DataTypes.py"
True
False
False
b is not greater than a
shtlp_0061@SHTLP0061:~/Python Basics$
```

Math functions:

The math module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using **import math**.

1) math.sqrt() →return square root of a number

2) math.e → return Euler's number

3) math.pi → return pi value

4) math.ceil() → return ceil value of an integer

5) math.floor() → return floor value of integer

6) math.factorial

7) math.gcd()

8)math.fabs - return absolute value of an integer

9) math.pow()

10) math.log()(natural base), math.log2() (base 2), math.log3()(base 3)…

mathModule.py > ...

```python
import math

print(math.sqrt(36))

print(math.e)
print(math.pi)

x = 2.3

print(math.ceil(x))
print(math.floor(x))

x = 5
print(math.factorial(x))

a = 15
b = 3

print(math.gcd(a,b))

y = -5
print(math.fabs(y))

print(math.pow(2,3))

print ("The value of log 2 with base 3 is : ", end="")
print (math.log(2,3))

# returning the log2 of 16
print ("The value of log2 of 16 is : ", end="")
print (math.log2(16))

# returning the log10 of 10000
print ("The value of log10 of 10000 is : ", end="")
print (math.log10(10000))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
6.0
2.718281828459045
3.141592653589793
3
2
120
3
5.0
8.0
The value of log 2 with base 3 is : 0.6309297535714574
The value of log2 of 16 is : 4.0
The value of log10 of 10000 is : 4.0
shtlp_0061@SHTLP0061:~/Python Basics$
```

Operator precedence and associativity:

Operator	Description	Associativity
( )	Parentheses	left-to-right
**	Exponent	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
is, is not in, not in	Identity Membership operators	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
not	Logical NOT	right-to-left
and	Logical AND	left-to-right
or	Logical OR	left-to-right
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left

Variables:

- A variable is created the moment you first assign a value to it.
- Variables do not need to be declared with any particular *type*, and can even change type after they have been set.
- If you want to specify the data type of a variable, this can be done with casting.

ex →

x = str(3)

```
y = int(3)
z = float(3)
```

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python Keywords.

variable assignment and output variables:

variable.py > ...

```python
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

# variable assignment
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
print('\n')
x = y = z = "Orange"
print(x)
print(y)
print(z)
print('\n')
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/variable.py"
Orange
Banana
Cherry


Orange
Orange
Orange


apple
banana
cherry
shtlp_0061@SHTLP0061:~/Python Basics$
```

variable.py > ...

```python

# output variables
x = "Python is awesome"
print(x)

x = "Python"
y = "is"
z = "awesome"
print(x, y, z)

x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/variable.py"
Python is awesome
Python is awesome
Python is awesome
shtlp_0061@SHTLP0061:~/Python Basics$
```

Global variables:

variable.py > ...

```python

# global variables

x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()



x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/variable.py"
Python is awesome
Python is fantastic
Python is awesome
shtlp_0061@SHTLP0061:~/Python Basics$
```

Expressions vs Statement:

- An expression is a combination of operators and operands that is interpreted to produce some other value.

1. Constant Expressions: These are the expressions that have constant values only.

2. Arithmetic Expressions: An arithmetic expression is a combination of numeric values, operators, and sometimes parenthesis.

operators → **+,-,*,/,//,%,****

3. Integral Expressions: These are the kind of expressions that produce only integer results after all computations and type conversions.

```python
expressions.py > ...

print("constant expression value:")
print(x)
print('\n')


# Arithmetic Expressions
x = 100
y = 50

add = x + y
sub = x - y
pro = x * y
div = x / y
print("arithmetic expression value:")
print(add)
print(sub)
print(pro)
print(div)
print('\n')

# Integral Expressions
a = 13
b = 12.0

c = a + int(b)
print("Integral Expression:")
print(c)
print('\n')
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/expressions.py"
constant expression value:
16.3


arithmetic expression value:
150
50
5000
2.0


Integral Expression:
25


shtlp_0061@SHTLP0061:~/Python Basics$
```

4. Floating Expressions: These are the kind of expressions which produce floating point numbers as result after all computations and type conversions.

5. Relational Expressions: In these types of expressions, arithmetic expressions are written on both sides of relational operator (> , < , >= , <=). Those arithmetic expressions are evaluated first, and then compared as per relational operator and produce a boolean output in the end.

6. Logical Expressions: These are kinds of expressions that result in either true or false.

7. Bitwise Expressions: These are the kind of expressions in which computations are performed at bit level.

8. Combinational Expressions: We can also use different types of expressions in a single expression.

expressions.py > ...

```python
#floating expression

a = 5
b = 3
print(a/b)

# boolean expression (>,<,<=,>=)
a = 1
b = 2
print(a > b)

# logical expression

a = (10 == 10)
b = (5>4)
c = 0

print(a and b)
print(a or c)

# Bitwise Expressions
a = 12

x = a >> 2
# right shift by 2
y = a << 1
#  left shift by 1

print(x, y)


# Combinational Expressions
a = 16
b = 12

c = a + (b >> 1)
print(c)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/expressions.py"
1.6666666666666667
False
True
True
3 24
22
shtlp_0061@SHTLP0061:~/Python Basics$
```

- A Python statement is an instruction that the Python interpreter can execute. There are different types of statements in Python language as Assignment statements, Conditional statements, Looping statements, etc.

1) if, elseif, else

statements.py > ...

```python
# if else if and else statment
if True:
    print('its true')

if 3>2:
    print("true")
else:
    print("false")


x = 'bank'

if x == 'not bank':
    print("cool")
elif x == 'hello':
    print("hello")
else:
    print("bank")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/statements.py"
its true
true
bank
shtlp_0061@SHTLP0061:~/Python Basics$
```

2) for loop

statements.py > ...

```python


# for loop
mylist = [1,2,3,4,5,6,7,8,9,10]

for i in mylist:
    print(i)

# check for even
for i in mylist:
    if i%2 == 0:
        print(f'even : {i}')
    else:
        print(f'odd : {i}')


mylist = [(1,2,3),(4,5,6),(7,8,9)]
for i in mylist:
    print(i)


for a,b,c in mylist:
    print(c)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/statements.py"
1
2
3
4
5
6
7
8
9
10
odd : 1
even : 2
odd : 3
even : 4
```

3) while loop

statements.py > ...

```python
# while loop
x = 0
while x<5:
    print(f'{x} is less than 5')
    x +=1
else:
    print("x is not less than 5")

# break continue and pass in loop

for i in "hello":
    # comment
    pass
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/statements.py"
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
x is not less than 5
shtlp_0061@SHTLP0061:~/Python Basics$
```

Augmented assignment operator:

+=

-=

/=

***=**

//=

%=

****=**

&=

|=

^=

<<=

>>=

Type Conversion:

1. Implicit Type Conversion → automatically converts one data type to another without any user involvement.

```
typeConversion.py > ...
x = 10

print("x is of type:",type(x))

y = 10.6
print("y is of type:",type(y))

z = x + y

print(z)
print("z is of type:",type(z))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Ba
x is of type: <class 'int'>
y is of type: <class 'float'>
20.6
z is of type: <class 'float'>
shtlp_0061@SHTLP0061:~/Python Basics$
```

2. Explicit Type Conversion → manually changed by the user as per their requirement.

```
typeConversion.py > ...

# explicit type conversion
s = "10010"

# printing string converting to int base 2
c = int(s,2)
print ("After converting to integer base 2 : ", end="")
print (c)

# printing string converting to float
e = float(s)
print ("After converting to float : ", end="")
print (e)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/typeConversion.py"
After converting to integer base 2 : 18
After converting to float : 10010.0
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/typeConversion.py"
```

Mutable vs Immutable Objects in Python:

Immutable: an immutable object can't be changed after it is created. ex - tuple, string ..

mutable: an mutable object can be changed after it is created. ex - list …

Built in functions in Python:

Function	Description
abs()	Returns the absolute value of a number
all()	Returns True if all items in an iterable object are true
any()	Returns True if any item in an iterable object is true
ascii()	Returns a readable version of an object. Replaces none-ascii characters with escape character
bin()	Returns the binary version of a number
bool()	Returns the boolean value of the specified object

bytearray()	Returns an array of bytes
bytes()	Returns a bytes object
callable()	Returns True if the specified object is callable, otherwise False
chr()	Returns a character from the specified Unicode code.
classmethod()	Converts a method into a class method
compile()	Returns the specified source as an object, ready to be executed
complex()	Returns a complex number

Functions:

test.py functions.py

Python Baisics 4 > functions.py > ...

```python
# function creation
def my_function():
  print("Hello from a function")

# function calling
my_function()


# function arguments and parameters
def my_function(fname):
  print("Hello "+fname)

my_function("Emil")
my_function("Tobias")
my_function("Linus")


# multiple parameters
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("riya", "rana")


# Arbitrary Arguments, *args
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
shtlp_0061@SHTLP0061:~/Python Basics$ /bin/python3 "/home/shtlp_0061/Python Basics/Python Baisics 4/fun
Hello from a function
Hello Emil
Hello Tobias
Hello Linus
riya rana
shtlp_0061@SHTLP0061:~/Python Basics$
```

Arguments/parameters:

Arbitrary Arguments, *args →

If you do not know how many arguments that will be passed into your function, add a `*` before the parameter name in the function definition.

This way the function will receive a *tuple* of arguments, and can access the items accordingly

Keyword argument →

You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.

Arbitrary Keyword Arguments, **kwargs →

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: `**` before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly