Python

python was created by Guido Van Rossum 1991. it is used for web development , software development etc.

how to install python

In [1]:
```
pip install python
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Could not find a version that satisfies the requirement python (from v
ersions: none)
ERROR: No matching distribution found for python
```

python keywords

In [2]:
```
help("%killbgscriptseywords")
```

```
No Python documentation found for '%killbgscriptseywords'.
Use help() to get the interactive help utility.
Use help(str) for help on the str class.
```

Variables

Python has no command for declaring a variable.

Rules to declare a variable

- Variable name should not start with a number.
- There should not be gaps while declaring a variable.
- Variable are case sensitive i.e, **DATA** and **data** both are different

In [3]:
```
x = 54
y = "Riya Wagh"
print(x)
print(y)
```

```
54
Riya Wagh
```

Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement, which is also known as multiple assignments.

In [4]:
```
x=y=z=50
print(x)
print(y)
print(z)
```

```
50
50
50
```

In [5]:
```
a,b,c = 5,10,15
print(a)
print (b)
print (c)
```

```
5
10
15
```

Delete a variable

We can delete the variable using the del keyword. The syntax is given below.

In [6]:
```
print(a)
del a
print(a)
```

```
5

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7624\3563035178.py in <module>
      1 print(a)
      2 del a
----> 3 print(a)

NameError: name 'a' is not defined
```

Data Type

- **Text Type**: str
- **Numeric Types**: int, float, complex
- **Sequence Types**: list, tuple, range
- **Mapping Type**: dict
- **Set Types**: set, frozenset

- **Boolean Type**: bool
- **Binary Types**: bytes, bytearray, memoryview

Numarical Data Type

Int

- Integer value can be any length such as integers 10, 2, 29, -20, -150 etc.
- Python has no restriction on the length of an integer. Its value belongs to int

Float

- Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc.
- It is accurate upto 15 decimal points.

complex

- A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively.
- The complex numbers like 2.14j, 2.0 + 2.3j, etc

Concatination Between Two Different Types

Text data type

String

- The string can be defined as the sequence of characters represented in the quotation marks.
- we can use single, double, or triple quotes to define a string.

In the case of string handling, the operator + is used to concatenate two strings as the operation "hello"+" python" returns "hello python".

In [ ]:
```
int("1") + 1
```

In [ ]:
```
"riya " + "wagh"
```

In [7]:
```
1+1
```

Out[7]:
```
2
```

The operator * is known as a repetition operator as the operation "Python" *2 returns 'Python Python'.

In [8]:
```
"1"*100
```

Out[8]:
```
'1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111'
```

In [9]:
```
a="My name is riya"
```

Upper Case

In [10]:
```
print(a.upper())
```

```
MY NAME IS RIYA
```

Lower Case

In [11]:
```
print(a.lower())
```

```
my name is riya
```

Remove Whitespace

The strip() method removes any whitespace from the beginning or the end: a = " my name is riya " print(a) print(a.strip())

In [12]:
```
a = "  my name is riya "
print(a)
print(a.strip())
```

```
  my name is riya 
my name is riya
```

Replace String

In [13]:
```
print(a.replace("r","R"))
```

```
  my name is Riya
```

Split String

The split() method returns a list where the text between the specified separator becomes the list items.

In [14]:
```
print(a.split(" "))
```

```
['', '', 'my', 'name', 'is', 'riya', '']
```

String Concatenation

In [15]:
```
b = "i want to became data science"
print(a)
print(b)
print(a+b)
```

```
  my name is riya
i want to became data science
  my name is riya i want to became data science
```

String Format

In [16]:
```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7624\982055754.py in <module>
      1 age = 36
----> 2 txt = "My name is John, I am " + age
      3 print(txt)

TypeError: can only concatenate str (not "int") to str
```

In [17]:
```
age = 36
txt = "My name is John, I am {}"
print(txt.format(age))
```

```
My name is John, I am 36
```

In [18]:
```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

Escape Character

```
Code    Result  Try it
\'      Single Quote
\\      Backslash
\n      New Line
\r      Carriage Return
\t      Tab
\b      Backspace
\f      Form Feed
```

```
\ooo    Octal value
\xhh    Hex value
```

In [19]:
```
txt = 'It\'s alright.'
print(txt)
txt = 'It \\ is alright.'
print(txt)
txt = 'It \n is alright.'
print(txt)
txt = 'It   is \r alright.'
print(txt)
txt = 'It   is \t alright.'
print(txt)
txt = 'It   is \b alright.'
print(txt)
txt = 'It   is \f alright.'
print(txt)
txt = "\110\145\154\154\157"
txt
txt = "\x48\x65\x6c\x6c\x6f"
txt
```

```
It's alright.
It \ is alright.
It
 is alright.
 alright.
It   is          alright.
It   is alright.
It   is  alright.
```

Out[19]:
```
'Hello'
```

String Methods

capitalize() Converts the first character to upper case

casefold() Converts string into lower case

center() Returns a centered string

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

encode() Returns an encoded version of the string

endswith() Returns true if the string ends with the specified value

expandtabs() Sets the tab size of the string

find() Searches the string for a specified value and returns the position of where it was found

format() Formats specified values in a string

format_map() Formats specified values in a string

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

isalnum() Returns True if all characters in the string are alphanumeric

isalpha() Returns True if all characters in the string are in the alphabet

isdecimal() Returns True if all characters in the string are decimals

isdigit() Returns True if all characters in the string are digits

isidentifier() Returns True if the string is an identifier

islower() Returns True if all characters in the string are lower case

isnumeric() Returns True if all characters in the string are numeric

isprintable() Returns True if all characters in the string are printable

isspace() Returns True if all characters in the string are whitespaces

istitle() Returns True if the string follows the rules of a title

isupper() Returns True if all characters in the string are upper case

join() Joins the elements of an iterable to the end of the string

ljust() Returns a left justified version of the string

lower() Converts a string into lower case

lstrip() Returns a left trim version of the string

maketrans() Returns a translation table to be used in translations

partition() Returns a tuple where the string is parted into three parts

replace() Returns a string where a specified value is replaced with a specified value

rfind() Searches the string for a specified value and returns the last position of where it was found

rindex() Searches the string for a specified value and returns the last position of where it was found

rjust() Returns a right justified version of the string

rpartition() Returns a tuple where the string is parted into three parts

rsplit() Splits the string at the specified separator, and returns a list

rstrip() Returns a right trim version of the string

split() Splits the string at the specified separator, and returns a list

splitlines() Splits the string at line breaks and returns a list

startswith() Returns true if the string starts with the specified value

strip() Returns a trimmed version of the string

swapcase() Swaps cases, lower case becomes upper case and vice versa

title() Converts the first character of each word to upper case

translate() Returns a translated string

upper() Converts a string into upper case

zfill() Fills the string with a specified number of 0 values at the beginning

Sequence Data Types

List

- Python Lists are similar to arrays in C.
- However, the list can contain data of different types.
- The items stored in the list are separated with a comma (,) and enclosed within square brackets [].
- We can use slice [:] operators to access the data of the list.
- The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

In [20]:
```
list1  = [1 ,2.5, "hi", "Riya"]
#Checking type of given list
print(type(list1))

#Printing the list1
print (list1)

# List slicing
print (list1[3:])

# List slicing
print (list1[0:2])

# List Concatenation using + operator
print (list1 + list1)

# List repetation using * operator
print (list1 * 3)
```

```
<class 'list'>
[1, 2.5, 'hi', 'Riya']
['Riya']
[1, 2.5]
[1, 2.5, 'hi', 'Riya', 1, 2.5, 'hi', 'Riya']
[1, 2.5, 'hi', 'Riya', 1, 2.5, 'hi', 'Riya', 1, 2.5, 'hi', 'Riya']
```

In [21]:
```
# List Length
a = ["apple" , "banana" , "cherry"]
print(a)
print(len(a))
print(type(a))
```

```
['apple', 'banana', 'cherry']
3
<class 'list'>
```

Acess list item

In [22]:
```
b=list(("apple","mango","banana"))
print(b)
print(b[1])
print(b[-1])
print(b[0:4])
print(b[1:3])
print(b[:1])
print(b[1:])
print(b[::-1])
for i in b:
    print(i)
```

```
['apple', 'mango', 'banana']
mango
banana
['apple', 'mango', 'banana']
['mango', 'banana']
['apple']
['mango', 'banana']
['banana', 'mango', 'apple']
apple
mango
banana
```

Change Item Value

In [23]:
```
print(b)
b[1] = "kiwi"
print(b)
```

```
['apple', 'mango', 'banana']
['apple', 'kiwi', 'banana']
```

In [24]:
```
print(b)
b[1:3] = [ "kiwi" , "watermallon" , "orange" ]
print(b)
```

```
['apple', 'kiwi', 'banana']
['apple', 'kiwi', 'watermallon', 'orange']
```

Append

In [25]:
```
a.append("orange")
print(a)
```

```
['apple', 'banana', 'cherry', 'orange']
```

Insert

In [26]:
```
print(a)
a.insert(0,"pea")
print(a)
```

```
['apple', 'banana', 'cherry', 'orange']
['pea', 'apple', 'banana', 'cherry', 'orange']
```

Extend

In [27]:
```
print(a)
print(b)
a.extend(b)
a
```

```
['pea', 'apple', 'banana', 'cherry', 'orange']
['apple', 'kiwi', 'watermallon', 'orange']
```

Out[27]:
```
['pea',
 'apple',
 'banana',
 'cherry',
 'orange',
 'apple',
 'kiwi',
 'watermallon',
 'orange']
```

Remove item from list

In [28]:
```
print(a)
a.remove("apple")
print(a)
```

```
['pea', 'apple', 'banana', 'cherry', 'orange', 'apple', 'kiwi', 'watermallo
n', 'orange']
['pea', 'banana', 'cherry', 'orange', 'apple', 'kiwi', 'watermallon', 'orang
e']
```

In [29]:
```
del a
a
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7624\3125231900.py in <module>
      1 del a
----> 2 a

NameError: name 'a' is not defined
```

In [30]:
```
b.clear()
b
```

Out[30]:
```
[]
```

In [31]:
```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
    if "a" in x:
        newlist.append(x)

print(newlist)
```

```
['apple', 'banana', 'mango']
```

In [32]:
```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

```
['apple', 'banana', 'mango']
```

In [33]:
```
newlist = [x for x in range(10)]
newlist
```

Out[33]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Sorting

In [34]:
```
fruits.sort()
fruits
```

Out[34]:
```
['apple', 'banana', 'cherry', 'kiwi', 'mango']
```

In [35]:
```
fruits.sort(reverse = True)
fruits
```

Out[35]:
```
['mango', 'kiwi', 'cherry', 'banana', 'apple']
```

Customize Sort Function

In [36]:
```
def myfunc(n):
    return abs(n - 50)

thislist = [100, 50, 65, 82, 23]
# 50 0 15 32 27
# 0 15 27 32 50
print(thislist)
thislist.sort(key = myfunc)
print(thislist)
```

```
[100, 50, 65, 82, 23]
[50, 65, 23, 82, 100]
```

In [37]:
```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
thislist.sort(key = str.lower)
print(thislist)
thislist.reverse()
print(thislist)
```

```
['Kiwi', 'Orange', 'banana', 'cherry']
['banana', 'cherry', 'Kiwi', 'Orange']
['Orange', 'Kiwi', 'cherry', 'banana']
```

In [38]:
```
number = [1,5,4,8,1,8,5,4,1,8,4,8,8,9,61,7,2,3,80,10,51,10,2,10,21,10,10,2,20,5
number.count(1)
```

Out[38]:
```
7
```

```
append()    Adds an element at the end of the list
clear()     Removes all the elements from the list
copy()      Returns a copy of the list
count()     Returns the number of elements with the specified value
extend()    Add the elements of a list (or any iterable), to the end of the
current list
index()     Returns the index of the first element with the specified value
insert()    Adds an element at the specified position
pop()       Removes the element at the specified position
remove()    Removes the item with the specified value
reverse()   Reverses the order of the list
sort()      Sorts the list
```

Tuple

- A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().
- A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

In [39]:
```
tup  = (1,2,"hi", "Riya", 2)
# Checking type of tup
print (type(tup))

#Printing the tuple
print (tup)

# Tuple slicing
print (tup[1:])
print (tup[0:1])

# Tuple concatenation using + operator
print (tup + tup)

# Tuple repatation using * operator
print (tup * 3)

# Adding value to tup. It will throw an error.
t[2] = "hi"
```

```
<class 'tuple'>
(1, 2, 'hi', 'Riya', 2)
(2, 'hi', 'Riya', 2)
(1,)
(1, 2, 'hi', 'Riya', 2, 1, 2, 'hi', 'Riya', 2)
(1, 2, 'hi', 'Riya', 2, 1, 2, 'hi', 'Riya', 2, 1, 2, 'hi', 'Riya', 2)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7624\3721312975.py in <module>
     17
     18 # Adding value to tup. It will throw an error.
---> 19 t[2] = "hi"

NameError: name 't' is not defined
```

In [40]:
```
fruits = ("apple", "banana", "cherry")
print(fruits)
print(len(fruits))
print(type(fruits))
```

```
('apple', 'banana', 'cherry')
3
<class 'tuple'>
```

Acess touple

In [41]:
```
print(fruits[:])
print(fruits[1])
print(fruits[-1])
print(fruits[1:6])
print(fruits[:6])
print(fruits[1:])
print(fruits[::-1])
```

```
('apple', 'banana', 'cherry')
banana
cherry
('banana', 'cherry')
('apple', 'banana', 'cherry')
('banana', 'cherry')
('cherry', 'banana', 'apple')
```

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

In [42]:
```
print(fruits)
fruits = list(fruits)
print(fruits)
fruits[1] = "kiwi"
print(fruits)
fruits = tuple(fruits)
print(fruits)
```

```
('apple', 'banana', 'cherry')
['apple', 'banana', 'cherry']
['apple', 'kiwi', 'cherry']
('apple', 'kiwi', 'cherry')
```

In [43]:
```
del fruits
print(fruits)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7624\753144438.py in <module>
      1 del fruits
----> 2 print(fruits)

NameError: name 'fruits' is not defined
```

Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

In [44]:
```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

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

```
apple
banana
cherry
```

Using Asterisk*

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:

In [45]:
```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

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

```
apple
banana
['cherry', 'strawberry', 'raspberry']
```

In [46]:
```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)
```

```
apple
['mango', 'papaya', 'pineapple']
cherry
```

Tuples Join

In [47]:
```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

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

```
('a', 'b', 'c', 1, 2, 3)
```

In [48]:
```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

In [49]:
```
 tuple2.count(1)
```

Out[49]:
```
1
```

In [50]:
```
tuple2.index(2)
```

Out[50]:
```
1
```

```
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
```

Range

In [51]:
```
x = range(10)
print(type(x))
print(x)
```

```
<class 'range'>
range(0, 10)
```

Random Number

In [52]:
```
import random

print(random.randrange(1, 10))
```

```
2
```

Mapping Data Type:

dicttionary

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

In [53]:
```
d = {"Name":'Riya', "Class":'M.Sc Computer science', "Course":"Data science mas

# Printing dictionary
print (d)

# Accesing value using keys
print("1st name is "+d["lass"])
print("2nd name is "+ d["Language"])

print (d.keys())
print (d.values())
```

```
{'Name': 'Riya', 'Class': 'M.Sc Computer science', 'Course': 'Data science ma
sters', 'Language': 'Python'}

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7624\2139094143.py in <module>
      5
      6 # Accesing value using keys
----> 7 print("1st name is "+d["lass"])
      8 print("2nd name is "+ d["Language"])
      9

KeyError: 'lass'
```

In [54]:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

In [55]:
```
print(len(thisdict))
```

```
3
```

In [56]:
```
print(type(thisdict))
```

```
<class 'dict'>
```

In [57]:
```
print(thisdict["model"])
```

```
Mustang
```

Accessing Items

In [58]:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["model"])
print(thisdict["model"])
print(thisdict.get("model"))
```

```
Mustang
Mustang
Mustang
```

In [59]:
```
print(thisdict.keys())
print(thisdict.values())
```

```
dict_keys(['brand', 'model', 'year'])
dict_values(['Ford', 'Mustang', 1964])
```

update values

In [60]:
```
thisdict.update({"year":2020})
print(thisdict)
thisdict["color"] = "red"
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020, 'color': 'red'}
```

Removing Items

In [61]:
```
thisdict.pop("model")
thisdict.popitem()
print(thisdict)
del thisdict["brand"]
print(thisdict)
thisdict.clear()
print(thisdict)
del thisdict
print(thisdict)
```

```
{'brand': 'Ford', 'year': 2020}
{'year': 2020}
{}

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7624\4131858221.py in <module>
      7 print(thisdict)
      8 del thisdict
----> 9 print(thisdict)

NameError: name 'thisdict' is not defined
```

In [62]:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
    print(x)
for x in thisdict:
    print(thisdict[x])
for x in thisdict.values():
    print(x)
for x in thisdict.keys():
    print(x)
for x, y in thisdict.items():
    print(x, y)
```

```
brand
model
year
Ford
Mustang
1964
Ford
Mustang
1964
brand
model
year
brand Ford
model Mustang
year 1964
```

Method Description

- **clear()** Removes all the elements from the dictionary
- **copy()** Returns a copy of the dictionary
- **fromkeys()** Returns a dictionary with the specified keys and value
- **get()** Returns the value of the specified key
- **items()** Returns a list containing a tuple for each key value pair
- **keys()** Returns a list containing the dictionary's keys
- **pop()** Removes the element with the specified key
- **popitem()** Removes the last inserted key-value pair
- **setdefault()** Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
- **update()** Updates the dictionary with the specified key-value pairs
- **values()** Returns a list of all the values in the dictionary

Boolean Data type

boolean

- Boolean type provides two built-in values, True and False.
- These values are used to determine the given statement true or false.
- It denotes by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F'

In [63]:
```
print(type(True))
print(type(False))
```

```
<class 'bool'>
<class 'bool'>
```

In [64]:
```
print(6<9)
print(6>9)
print(6==9)
print(6<=9)
print(6>=9)
```

```
True
False
False
True
False
```

In [65]:
```
bool("hello")
```

Out[65]:
```
True
```

In [66]:
```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

```
False
False
False
False
False
False
False
```

Set Data Type

- Python Set is the unordered collection of the data type.
- It is iterable, mutable(can modify after creation), and has unique elements.
- In set, the order of the elements is undefined; it may return the changed sequence of the element.
- The set is created by using a built-in function set(), or a sequence of elements is passed in the curly braces and separated by the comma.
- It can contain various types of values.

In [67]:
```
# Creating Empty set
set1 = set()

set2 = {'Riya', 2, 3,'Wagh'}

#Printing Set value
print(set2)

# Adding element to the set

set2.add(10)
print(set2)

#Removing element from the set
set2.remove(2)
print(set2)
```

```
{2, 3, 'Riya', 'Wagh'}
{2, 3, 'Riya', 10, 'Wagh'}
{3, 'Riya', 10, 'Wagh'}
```

how check data type

type() is use to check data type in python

In [68]:
```
x = 54
y = "Riya Wagh"
print(x)
print(y)
print(type(x))
print(type(y))
```

```
54
Riya Wagh
<class 'int'>
<class 'str'>
```

In [69]:
```
a={1,5,8,8,6,5,4,7,6,3,1,4,6,2,7,8}
print(a)
print(type(a))
print(len(a))
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
<class 'set'>
8
```

In [70]:
```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

```
True
```

Add item in set

In [71]:
```
thisset.add("orange")

print(thisset)
```

```
{'apple', 'cherry', 'orange', 'banana'}
```

Add set in set

In [72]:
```
myset = {"kiwi", "orange"}

thisset.update(myset)

print(thisset)
```

```
{'apple', 'kiwi', 'banana', 'cherry', 'orange'}
```

Add Any Iterable

In [73]:
```
mylist = ["watermalon", "guvava"]

thisset.update(mylist)

print(thisset)
```

```
{'apple', 'watermalon', 'kiwi', 'banana', 'guvava', 'cherry', 'orange'}
```

In [74]:
```
mytuple = {"grapes"}

thisset.update(mytuple)

print(thisset)
```

```
{'apple', 'watermalon', 'grapes', 'kiwi', 'banana', 'guvava', 'cherry', 'oran
ge'}
```

In [75]:
```
mydict = {"fruit1" : "bluebares","fruit2" : "papaya"}

thisset.update(mydict)

print(thisset)
```

```
{'banana', 'cherry', 'kiwi', 'orange', 'apple', 'watermalon', 'fruit2', 'frui
t1', 'guvava', 'grapes'}
```

Python - Remove Set Items

In [76]:
```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

```
{'apple', 'cherry'}
```

In [77]:
```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)

thisset.pop()

print(thisset)

thisset.clear()

thisset

del thisset

print(thisset)
```

```
{'apple', 'cherry'}
{'cherry'}

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7624\1990391743.py in <module>
     15 del thisset
     16
---> 17 print(thisset)

NameError: name 'thisset' is not defined
```

In [78]:
```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

```
apple
{'cherry', 'banana'}
```

In [79]:
```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

```
set()
```

In [80]:
```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_7624\3077915618.py in <module>
      3 del thisset
      4
----> 5 print(thisset)

NameError: name 'thisset' is not defined
```

Python - Join Sets

In [81]:
```
# Join set
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

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

```
{1, 2, 3, 'c', 'a', 'b'}
```

In [82]:
```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

```
{1, 2, 3, 'c', 'a', 'b'}
```

In [83]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)

print(x)
```

```
{'apple'}
```

In [84]:
```
z=x.intersection(y)
print(z)
```

```
{'apple'}
```

In [85]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.symmetric_difference_update(y)

print(x)

x.symmetric_difference(y)

print(x)
```

```
{'microsoft', 'banana', 'google', 'cherry'}
{'microsoft', 'banana', 'google', 'cherry'}
```

- **Method** Description
- **add()** Adds an element to the set
- **clear()** Removes all the elements from the set
- **copy()** Returns a copy of the set
- **difference()** Returns a set containing the difference between two or more sets
- **difference_update()** Removes the items in this set that are also included in another, specified set
- **discard()** Remove the specified item
- **intersection()** Returns a set, that is the intersection of two other sets
- **intersection_update()** Removes the items in this set that are not present in other, specified set(s)
- **isdisjoint()** Returns whether two sets have a intersection or not
- **issubset()** Returns whether another set contains this set or not
- **issuperset()** Returns whether this set contains another set or not
- **pop()** Removes an element from the set
- **remove()** Removes the specified element
- **symmetric_difference()** Returns a set with the symmetric differences of two sets
- **symmetric_difference_update()** inserts the symmetric differences from this set and another
- **union()** Return a set containing the union of sets
- **update()** Update the set with the union of this set and others

Casting

Type Casting is the method to convert the variable data type into a certain data type

There can be two types of Type Casting in Python –

- Implicit Type Casting
- Explicit Type Casting

Inplicit type casting

In this, methods, Python converts data type into another data type automatically. In this process, users don't have to involve in this process.

In [86]:
```
x = 1
print(type(x))

y = 2.8
print(type(y))

z = "3"
print(type(z))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

Explicit Type Casting

Mainly in type casting can be done with these data type function:

- **Int()** : Int() function take float or string as an argument and return int type object.
- **float()** : float() function take int or string as an argument and return float type object.
- **str()** : str() function take float or int as an argument and return string type object.

In [87]:
```
x = int(1)   # x will be 1
print(type(x))
y = int(2.8) # y will be 2
print(type(y))
z = int("3") # z will be 3
print(type(z))
```

```
<class 'int'>
<class 'int'>
<class 'int'>
```

In [88]:
```
x = float(1)   # x will be 1
print(type(x))
y = float(2.8) # y will be 2
print(type(y))
z = float("3") # z will be 3
print(type(z))
```

```
<class 'float'>
<class 'float'>
<class 'float'>
```

In [89]:
```
x = str(1)   # x will be 1
print(type(x))
y = str(2.8) # y will be 2
print(type(y))
z = str("3") # z will be 3
print(type(z))
```

```
<class 'str'>
<class 'str'>
<class 'str'>
```

Formating

In [90]:
```
age = 21
print("my age is",age)
```

```
my age is 21
```

In [91]:
```
print("my age is {age}")
```

```
my age is {age}
```

In [92]:
```
print(f"my age is {age}")
print("my age is {}".format(age))
name=input("Enter your name")
print("My name is {}".format(name))
```

```
my age is 21
my age is 21
Enter your name
My name is
```

Operators

- Operators are used to perform operations on variables and values.
- Python divides the operators in the following groups:

- **Arithmetic operators**
- **Assignment operators**
- **Comparison operators**
- **Logical operators**
- **Identity operators**
- **Membership operators**
- **Bitwise operators**

```
+   Addition    x + y
-   Subtraction x - y
*   Multiplication  x * y
/   Division    x / y
%   Modulus x % y
```

```
**  Exponentiation  x ** y
//  Floor division  x // y
```

Assignment Operators

Assignment operators are used to assign values to variables:

```
=   x = 5   x = 5
+=  x += 3  x = x + 3
-=  x -= 3  x = x - 3
*=  x *= 3  x = x * 3
/=  x /= 3  x = x / 3
%=  x %= 3  x = x % 3
//= x //= 3 x = x // 3
**= x **= 3 x = x ** 3
&=  x &= 3  x = x & 3
|=  x |= 3  x = x | 3
^=  x ^= 3  x = x ^ 3
>>= x >>= 3 x = x >> 3
<<= x <<= 3 x = x << 3
```

Comparison Operators

Comparison operators are used to compare two values:

```
==  Equal   x == y
!=  Not equal   x != y
>   Greater than    x > y
<   Less than   x < y
>=  Greater than or equal to    x >= y
<=  Less than or equal to   x <= y
```

Logical Operators

Logical operators are used to combine conditional statements:

```
and     Returns True if both statements are true    x < 5 and  x < 10
or  Returns True if one of the statements is true   x < 5 or x < 4
not Reverse the result, returns False if the result is true not(x < 5 and x <
10)
```

Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

```
is  Returns True if both variables are the same object  x is y
is not  Returns True if both variables are not the same object  x is not y
```

** Membership Operators**

Membership operators are used to test if a sequence is presented in an object:

```
in  Returns True if a sequence with the specified value is present in the
object    x in y
not in  Returns True if a sequence with the specified value is not present in
the object    x not in y
```

Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

```
&       AND                     Sets each bit to 1 if both bits are 1
        x & y

|       OR                      Sets each bit to 1 if one of two bits is 1
        x | y

^       XOR                     Sets each bit to 1 if only one of two bits is
1      x ^ y

~       NOT                     Inverts all the bits
        ~x

<<      Zero fill left shift    Shift left by pushing zeros in from the right
       x << 2
                                and let the leftmost bits fall off

>>      Signed right shift      Shift right by pushing copies of the leftmost
       x >> 2
                                bit in from the left, and let the rightmost
bits
                                fall off
```

Operator Precedence

```
()            Parentheses

**            Exponentiation

+x  -x  ~x    Unary plus, unary minus, and bitwise NOT

*  /  //  %   Multiplication, division, floor division, and modulus
```

```
+  -          Addition and subtraction

<<  >>        Bitwise left and right shifts

&             Bitwise AND

^             Bitwise XOR

|             Bitwise OR

==  !=       Comparisons,
>   >=       identity, and
<  <=        membership operators
is  is not
in  not in

not           Logical NOT

and           AND

or            OR
```

Control Statement in python

logical conditions

```
Equals:                  a == b
Not Equals:              a != b
Less than:               a < b
Less than or equal to:   a <= b
Greater than:            a > b
Greater than or equal to: a >= b
```

Control Statement

- **if**
- **if-else**

In [93]:
```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

```
b is greater than a
```

In [94]:
```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

```
a and b are equal
```

In [95]:
```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

```
a is greater than b
```

In [96]:
```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

```
Both conditions are True
```

In [97]:
```
a = 200
b = 33
c = 500
if a > b or a > c:
    print("At least one of the conditions is True")
```

```
At least one of the conditions is True
```

In [98]:
```
a = 33
b = 200
if not a > b:
    print("a is NOT greater than b")
```

```
a is NOT greater than b
```

In [99]:
```
x = 41

if x > 10:
    print("Above ten,")
    if x > 20:
    print("and also above 20!")
    else:
    print("but not above 20.")
```

```
  File "C:\Users\riya wagh\AppData\Local\Temp\ipykernel_7624\2829210178.py",
line 6
    print("and also above 20!")
    ^
IndentationError: expected an indented block
```

In [100]:
```
a = 33
b = 200

if b > a:
    pass
```

Loops

- while loops
- for loops

In [101]:
```
i = 1
while i < 6:
    print(i)
    i += 1
```

```
1
2
3
4
5
```

In [102]:
```
i = 1
while i < 6:
 print(i)
 if i == 3:
     break
 i += 1
```

```
1
2
3
```

In [103]:
```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

```
1
2
4
5
6
```

In [104]:
```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

```
1
2
3
4
5
i is no longer less than 6
```

In [105]:
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

```
apple
banana
cherry
```

In [106]:
```
for x in "banana":
  print(x)
```

```
b
a
n
a
n
a
```

In [107]:
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
apple
banana
```

In [108]:
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

```
apple
```

In [109]:
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

```
apple
cherry
```

In [110]:
```
for x in range(6):
  print(x)
```

```
0
1
2
3
4
5
```

In [111]:
```
for x in range(2, 6):
  print(x)
```

```
2
3
4
5
```

In [112]:
```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```

In [113]:
```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

In [114]:
```
for x in [0, 1, 2]:
  pass
```

In [ ]:

In [ ]: