3/31/2022

Python Basics For Beginners

Author: Nitish k
College: RNS Institute of Technology

Python Basics

I. Datatypes In Python

a. Text Type:	str
b. Numeric Types:	int, float, complex
c. Sequence Types:	list, tuple, range
d. Mapping Type:	dict
e. Set Types:	set, frozenset
f. Boolean Type:	bool
g. Binary Types:	bytes, bytearray, memoryview

Note: Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

Eg:
```
import random
print(random.randrange(1, 10))
```

II. Strings in Python

Strings in python are surrounded by either single quotation marks, or double quotation marks. 'hello' is the same as "hello". You can display a string literal with the print() function:

Eg:
```
print("Hello")
print('Hello')
```

a. **Assign String to a Variable:** Assigning a string to a variable is done with the variable name followed by an equal sign and the string

Eg:
```
a = "Hello"
print(a)
```

b. **Multiline String:** You can assign a multiline string to a variable by using three quotes:

Eg:
```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

c. **Strings are Arrays:** Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

Eg:

a = "Hello, World!"
print(a[1])
Output: e

d. **String Length:** To get length of the string we use len() function
Eg: a = ‘Hello, World’
print(a)
Output: 13

e. **Check String:**
   i. To check if a string or phrase or character is present in a string, we can use the keyword in.
      Eg:
      Txt = “The best things in life are free!”
      print(“free” in txt)
      Output: True
   ii. To check if a string or phrase or character is not present in a string we use the keyword not in
      Eg:
      Txt = “The best things in life are free”
      print(“expensive” in txt)
      Output: False

f. **Slicing a String:** U can return a range of characters by using the slice syntax.
Eg:
b = “Hello World”
print(b[2:5])
Output: llo
   i. Slice from the start: By leaving out the start index, the range will start at the first character:
      Eg:
      b = “Hello, World!”
      print(b[:5])
      Output: Hello
   ii. Slice from the end: By leaving the end index, the range will go to the end
      Eg:
      b = “Hello World!”
      print(b[2:])
      Output: llo, World!

g. **Modify Strings:** There are a number of built in methods in python to modify strings

i. **Upper Case:** The upper() method returns the string in upper case
   Eg:
   a = “Hello, World!”
   print(a.upper())
   Output: HELLO, WORLD!

ii. **Lower Case:** The lower() method returns the string in lower case
   Eg:
   a = “Hello, World!”
   print(a.lower())
   Output: hello, world!

iii. **Remove Whitespace:** The strip() method is used to remove whitespace in a string
   Eg:
   a = “ Hello, World! ”
   print(a.strip())
   Output: Hello, World!

iv. **Replace String:** The replace() method replaces a string with another string:
   Eg:
   a = “Hello, World!”
   print(a.replace(“H”,”J”))
   Output: Jello, World!

v. **Split String:** The split() method returns a list
   Eg:
   a = “Hello, World!”
   print(a.split())
   Output: [‘Hello,’,’World’]

   We can specify a delimiter and the string will be splitted at that delimiter as shown
   Eg:
   a = “Hello, World”
   print(a.split(‘,’))
   Output: [‘Hello’,’ World’]

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

   Eg:

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

Eg:

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

vii. **String Methods:**

Method	Description
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

3. <u>Boolean Values:</u>

Most Values are True. Almost any value is evaluated to True if it has some sort of content. Any string is True, except empty strings. Any number is True, except 0. Any list, tuple, set, and dictionary are True, except empty ones.

Eg:

bool("abc")

bool(123)

bool(["apple", "cherry", "banana"])

All are evaluated to true

Some Values are False. In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

Eg:

bool(False)

bool(None)

```
bool(0)
bool("")
bool(())
bool([])
bool({})
```

All are evaluated to false

One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a __len__ function that returns 0 or False:

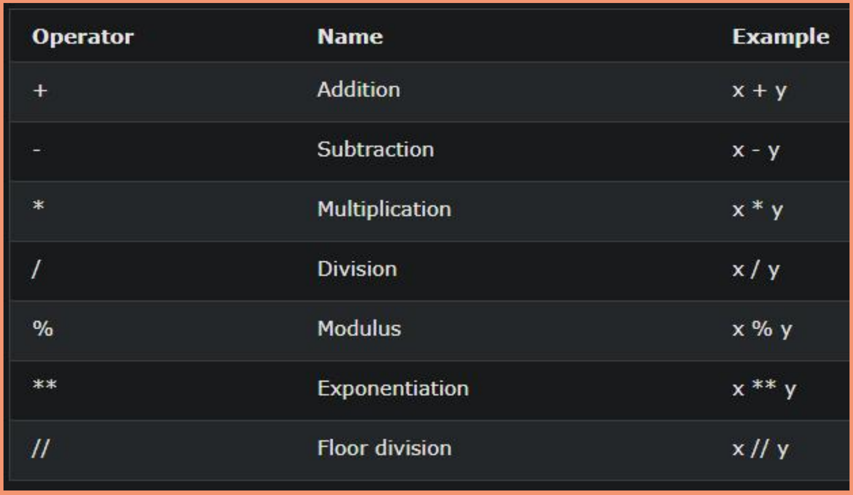
Eg:

```
class myclass():
  def __len__(self):
    return 0

myobj = myclass()
print(bool(myobj))
```

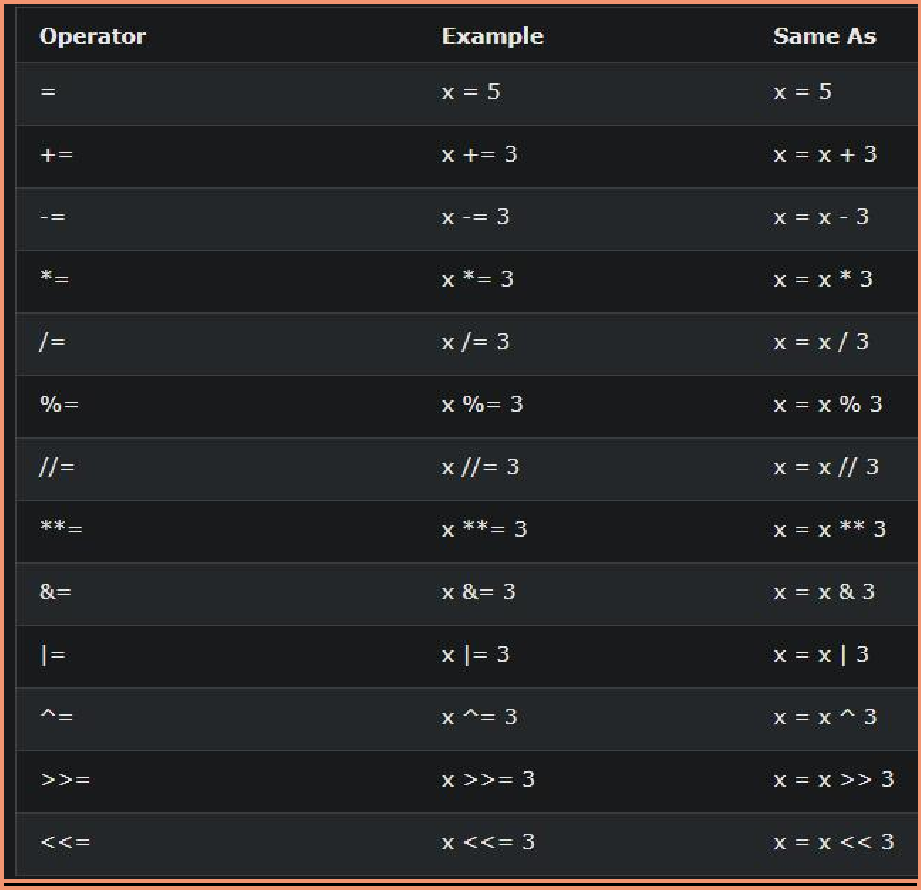
4. **Python Operators**:

   a. **Arithmetic Operators**:



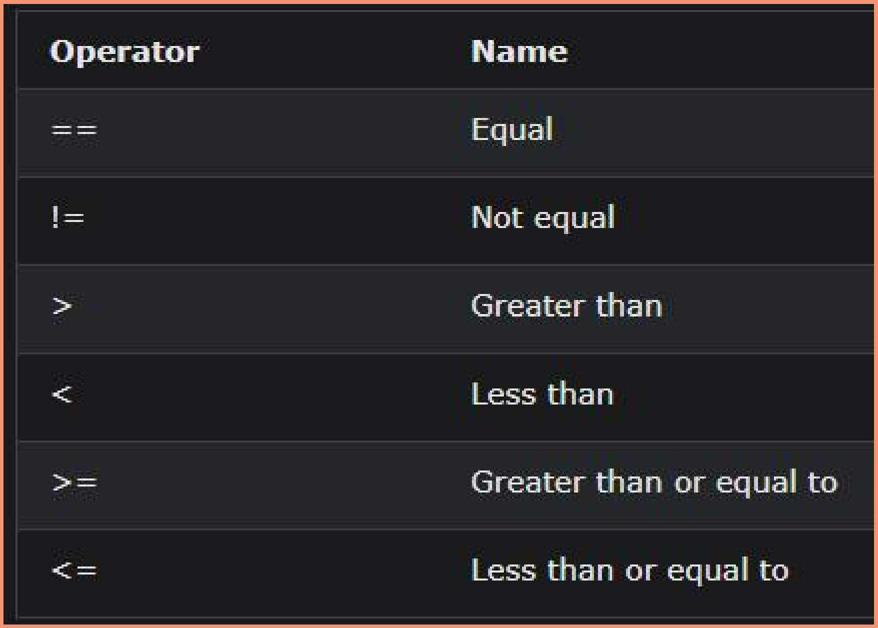
Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

   b. **Assignment Operator:**



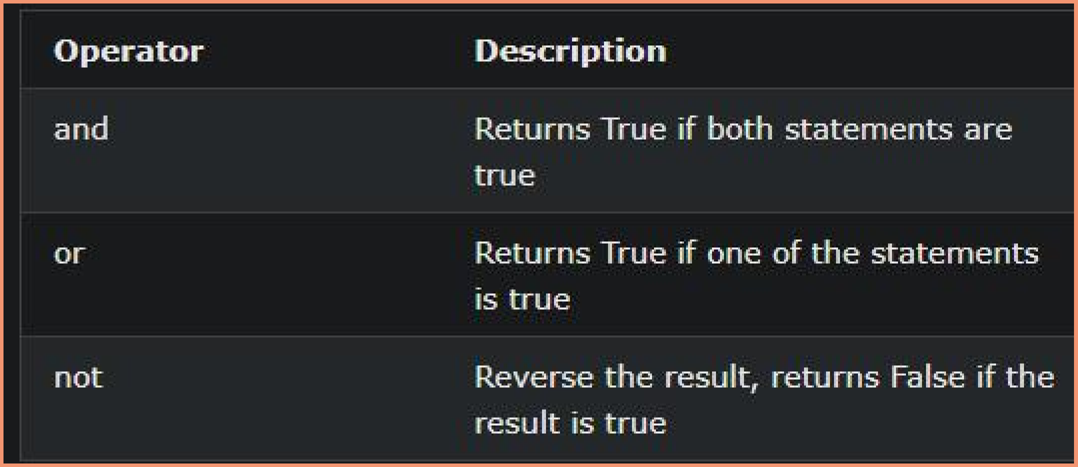
Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

c. **Comparison Operator:**



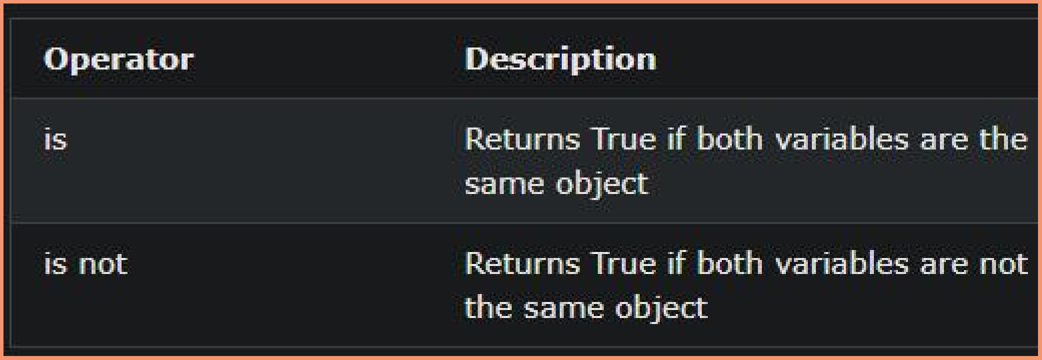
Operator	Name
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

d. **Logical Operator:**



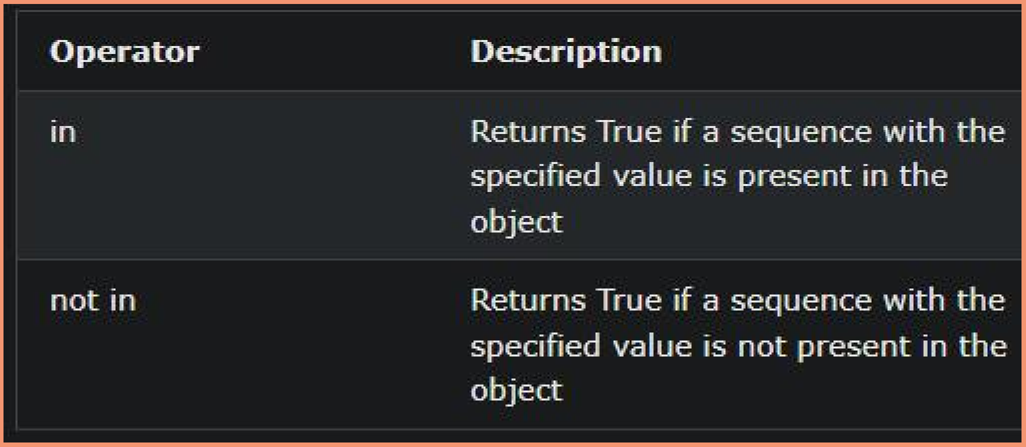
Operator	Description
and	Returns True if both statements are true
or	Returns True if one of the statements is true
not	Reverse the result, returns False if the result is true

e. **<u>Identity Operator:</u>**



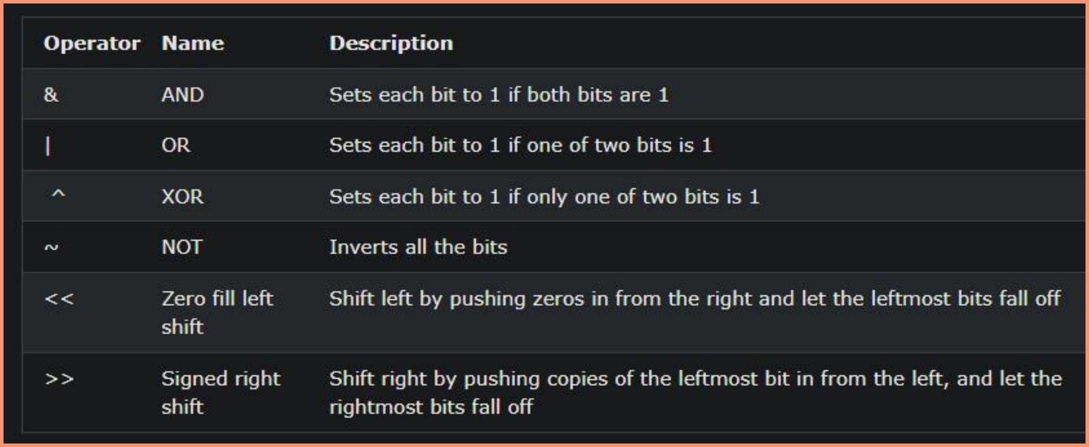
Operator	Description
is	Returns True if both variables are the same object
is not	Returns True if both variables are not the same object

f. **<u>Membership Operator:</u>**



Operator	Description
in	Returns True if a sequence with the specified value is present in the object
not in	Returns True if a sequence with the specified value is not present in the object

g. **<u>Bitwise Operator:</u>**



Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

III. <u>Data Structures in Python</u>

1) <u>List</u>

Lists are used to store multiple items in a single variable. Enclosed within squared brackets. List items are ordered, changeable, and allow duplicate values.

Eg) myList = [“dog”, “cat”,” mouse”]

print(myList)

To determine the length of the list we use the len() function

Eg: myList = [“dog”, “cat”,” mouse”]

print(len(myList))

<u>The list() constructor:</u>

It is possible to use the list() constructor when creating new list.

Eg:

myList = list((“dog”,”cat”,”banana”))

print(myList)

<u>Access Items:</u>

List items are indexed and you can access them by referring to the index number

Eg:

thislist = ["apple", "banana", "cherry"]

print(thislist[1])

<u>Range of Indexes:</u> You can specify a range of indexes by specifying where to start and where to end the range.

Eg:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Eg2:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

Eg3:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

Check if an item Exists

To determine if a specified item is present in a list use the in keyword:

Eg:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Change Item Value

Eg:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Change a Range of Item Values

Eg:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

1.1) Functions:

a. Extend() : list.extend(iterable)

    i. A=[1,2,3]
       B=[4,5,6]
       A.extend(B)
       Output: [1,2,3,4,5,6]

b. Index(): to list positions of elements

    i. A=[1,2,3,4]

```
A.index(3)
Output: 2
```

c. Append(): to add only 1 element
   i. A=[1,2,3,4]
      A.append(5)
      A
      Output:[1,2,3,4,5]

d. Insert Items: To insert a new list item, without replacing any of the existing values, we can use the insert() method.
   Eg:
   ```
   thislist = ["apple", "banana", "cherry"]
   thislist.insert(2, "watermelon")
   print(thislist)
   ```

e. Count: to count the number of elements
   i. A=[1,2,3,4,5,6,7,1,4,1]
      A.count(1)
      Output: 3
f. Remove(data): to remove element from particular position
   i. A=[1,2,3,4,5]
      A.remove(3)
      A
      Output: [1, 2, 4, 5]
g. Pop(index): to remove a particular value
   i. A=[1,2,3,4,5]
      A.pop(3)
      Output: 4
h. del: Removes the specified index
   Eg:
   ```
   thislist = ["apple", "banana", "cherry"]
   del thislist[0]
   print(thislist)
   ```

   Can also delete the list entirely
   Eg:
   ```
   thislist = ["apple", "banana", "cherry"]
   del thislist
   ```

i. clear(): The clear() method empties the list
   Eg:
   ```
   thislist = ["apple", "banana", "cherry"]
   thislist.clear()
   print(thislist)
   Output: []
   ```
j. sort(): used to sort the list in ascending order

   Eg:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

To sort the list in descending order

Eg:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

To print the list in reverse order

Eg:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.reverse()
print(thislist)
```

k. copy() : used to copy the contents of 1 list to another list

Eg;

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

1.2) Loop Lists: You can loop through lists using the for loop

Eg:

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

1.3) Loop Through the Index Numbers. Use the range() and len() functions to create a suitable iterable.

Eg:

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

1.4) Using a While loop: You can loop through the list items by using a while loop. Use the len() function to determine the length of the list, then start at 0 and loop your way through the list items by refering to their indexes. Remember to increase the index by 1 after each iteration.

Eg:

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

1.5) List comprehension: List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list. Example: Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Eg:

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

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

2) Row wise and Column wise indexing order

a. Example)2D list y= [12,3, [24,'list',46], [100,4500,56]]

```
y[2][1]
Output: 'list'
```

2. Tuple: Tuples are used to store multiple items in a single variable. Enclosed within round brackets. A tuple is a collection which is ordered and unchangeable.

Eg:

```
thistuple = (“apple”,” banana”,”cherry”)
print(thistuple)
```

2.1) Tuple Length: To determine tuple length.

Eg:

```
thistuple=(“apple”,”banana”,”cherry”)
print(len(thistuple))
```

2.2) tuple() constructor

It is possible to use the tuple() constructor to make a tuple.

Eg:

```
thistuple = tuple(("apple", "banana", "cherry"))
print(thistuple)
```

2.3) Access Tuple Items

By using index

Eg:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

2.4) Range of Indexes:

Case 1:

Eg:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

Case 2: By leaving out the start value, the range will start at the first item:
Eg:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

Case 3: By leaving out the end value, the range will go on to the end of the list:
Eg:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

2.5) To check if an Item is present or not: by using in and not in
Eg:

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

2.6) Update Tuples: tuples are immutable. But there are certain ways to change a tuple.

Method 1:
By changing the tuple to a list and then adding values and then again to convert it back to a tuple
Eg:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

Method 2:
You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

2.7) Remove items from a tuple: Since tuple is immutable u cannot remove values from a tuple but there are certain methods using which u can remove items from a tuple

Method 1:

Convert the tuple to a list
Eg:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
```

```
y.remove("apple")
thistuple = tuple(y)
```

Method 2

Delete the tuple completely using del keyword

Eg:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple)
```

2.8) **Unpack Tuple:**

When we create a tuple we assign values to it. This is called packing a tuple.

Eg: fruits = ("apple", "banana", "cherry")

Unpacking is reverse of packing.

Eg:

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

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

2.9) **Loop Through a Tuple**:

Using for loop

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Loop through index numbers

Using range() and len()

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

Using while loop

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

2.10) **Join two tuples:** To join two or more tuples using the + operator

Eg:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
```

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

Multiply Two tuples using *

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

2.11) <u>Methods In tuple:</u>

Method	Description
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

3) <u>Sets:</u>

Sets are used to store multiple items in a single variable. A set is a collection which is unordered, unchangeable*, and unindexed. Sets are written with curly brackets.

Eg:

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

Duplicates are not allowed

3.1) Length of a set:

To determine how many items a set has, use the len() function.

Eg:

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

3.2) type()

Eg:

```
myset = {"apple", "banana", "cherry"}
print(type(myset))
```

3.3) set() constructor:

Eg:

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets

print(thisset)
```

3.4) access items from a set

Eg:

```
thisset = {"apple", "banana", "cherry"}


for x in thisset:
  print(x)
```

3.5) in and not in

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

3.6) Add items:

To add one item to a set use the add() method.

Eg:

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

3.7) Add sets:

Using update() method

Eg:

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

3.8) Remove items from set:

remove()

Eg:

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

discard()

Eg:

```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```

pop() : used to remove the last item from the list

Eg:

```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)
print(thisset)
```

clear()

Eg:

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

del(): used to remove the set completely

Eg:

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

3.9) Loop through the set:

Using for loop

Eg:

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

3.10) Join Two Sets

union(): can be used to join two sets

Eg

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3):
```

intersection_update() : keeps elements present in both the sets

Eg:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
```

```
x.intersection_update(y)
print(x)
```

intersection(): returns a set that contains items from both the set

Eg:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.intersection(y)
print(z)
```

symmetric_difference(): returns a new set that contains elements which are not present in both set

Eg:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.symmetric_difference(y)
print(z)
```

4) <u>**Dictionary:**</u>

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.

```
Example: Month={1:'January',2:'February'}
      print(Month.keys())
      print(Month.values())
      print(Month.items())
```

```
Output: dict_keys([1, 2])
        dict_values(['January', 'February'])
        dict_items([(1, 'January'), (2, 'February')])
```

```
Example: States={'Id':[1,2,3,4],
         'Places':['Chennai','Pune','Mumbai','Delhi']}
         States
```

```
Output: {'Id': [1, 2, 3, 4], 'Places': ['Chennai', 'Pune', 'Mumbai', 'Delhi']}
```

```
Example: States['Id'][2]
```

```
Output: 3
```

```
Eg:     thisdict = {
        "brand": "Ford",
          "model": "Mustang",
          "year": 1964
        }
        print(thisdict)
```

Dictionary Items: are ordered, changeable, and does not allow duplicates. They can be referred by key names.

Eg:

Print the "brand" value of the dictionary:

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

4.1) Dictionary Length: To determine how many items a dictionary has, use the len() function:

Eg:

```
print(len(thisdict))
```

4.2) **type()**:

Eg:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(type(thisdict))
```

4.3) Access Items in a Dictionary:

U can access items in a dictionary using keys.

Eg: thisdict = {

```
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```

4.3.1) keys(): will return a list of all the keys in the dictionary.

Eg: `x = thisdict.keys()`

4.3.2) values(): will return a list of all the values in the dictionary.

Eg: `x = thisdict.values()`

4.3.3) items(): will return each item in a dictionary as tuples in a list

Eg: `x = thisdict.items()`

<u>4.4) Check if the key exists</u>: using in and not in

Eg:

```
thisdict = {
  "brand": "Ford",
 "model": "Mustang",
"year": 1964
}
if "model" in thisdict:
    print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

4.5) Update Dictionary: The update() method will update the dictionary with items from the given argument. The argument must be a dictionary, or an iterable object with key:value pair.

Eg:

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

4.6) Remove Items:

4.6.1) pop(): removes the items with specified key names:

Eg:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

4.6.2) del : removes the item with specified key name:

Eg:

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

4.6.3) clear() : empties the dictionary:

Eg:

```
thisdict = {
 "brand": "Ford",
  "model": "Mustang",
 "year": 1964
}
thisdict.clear()
print(thisdict)
```

<u>4.7) Loop through a Dictionary:</u>

Using for loop:

Eg:

```
for x in thisdict:
             print(x)
```

To print the values in a dictionary: using values() method

Eg:

```
for x in thisdict.values():
        print(x)
```

To print the keys() method to return the return the keys of a dictionary:

Eg:

```
for x in thisdict.keys():
    print(x)
```

To print both keys and values, by using the items() method:

Eg:

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

4.8) Copy Dictionaries:

Various ways to copy of a dictionary with copy() method:

Eg:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

4.9) Nested Dictionary

A dictionary can contain dictionaries, this is called nested dictionaries.

Eg:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

To concatenate a dictionary without using + operator is by using update method

```
Month1={1:'January',2:'February',3:'March'}
Month2={4:'April',5:'May',6:'June'}
Month1.update(Month2)
print(Month1)
Output: {1: 'January', 2: 'February', 3: 'March', 4: 'April', 5: 'May', 6: 'June'}
```

IV. <u>Python Conditions and If Statements</u>

Python supports the usual logical conditions from mathematics:

- 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

1) If Statements: An "if statement" is written by using the if keyword
Eg:
```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

2) Elif statements: The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Eg:
```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

3) Else statements: The else keyword catches anything which isn't caught by the preceding conditions.
Eg:
```
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")
```

4) Nested if:

Eg:

```
x = 41

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

Eg:

1) #Source Code for even or odd

```
 num = int(input("Enter a number"))
 if num % 2 == 0:
        print(num," is even")
else:
        print(num," is odd")
```

2) #Source Code for palindrome

```
word = input("Enter a word")
if(word[::-1]==word[:]):
        print(word," is a palindrome")
else:
        print(word," is not a palindrome")
```

3) #Source Code for multiplication for for loop

```
num = int(input("Enter a number for multiplication "))

limit = int(input(("Enter the limit ")))

for i in range(1,limit+1):

          print(num," * ",i," = ",(num*i))
```

V. <u>Python Loops:</u>

Python has two primitive loop commands: while loops, for loops

1) **<u>The while Loop</u>**:

With the while loop we can execute a set of statements as long as a condition is true.
Eg:
```
i = 1
while i < 6:
  print(i)
  i += 1
```

2) **The break statement:**
With the break statement we can stop the loop even if the while condition is true:
Eg:
```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

3) **The continue statement:**
With the continue statement we can stop the current iteration, and continue with the next:
Eg:
```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

4) **For loop:**
A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
Eg:
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

The Range() Function: To loop through a set of code a specified number of times, we can use the range() function, The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Eg:
```
for x in range(6):
  print(x)
```

VI. Python Functions:
A function is a block of code which only runs when it is called.

1) **Creating a Function:** In Python a function is defined using the def keyword

Eg:

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

2) **Calling a Function:** To call a function use the function name followed by parenthesis:

Eg:

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

3) **Arbitrary Arguments, *args:**
If the number of arguments is unknown, add a * before the parameter name:
Eg:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])
my_function("Emil", "Tobias", "Linus")
```

4) **Keyword Arguments:** You can also send arguments with the key = value syntax.
Eg:

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)
my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

5) **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:
Eg:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])
my_function(fname = "Tobias", lname = "Refsnes")
```

6) **Default Parameter Value:**
Eg:

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

7) **Passing a list as an argument:**

Eg:

```
def my_function(food):
  for x in food:
    print(x)
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

8) **Return Values:**

Eg:

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

9) **The pass Statement:**

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

Eg:

```
def myfunction():
  pass
```

VII. Python Lambda Functions

A lambda function is a small anonymous function. Can take any number of arguments, but can have one expression.

Syntax: lambda arguments : expression

Eg:

```
x = lambda a: a + 10
print(x(5))

y = lambda b: b*100
print(y(5))

z = lambda c: c-100
print(z(200))
```

Outputs:

```
15
500
100
```

Eg:

```
def myfunc(n):
  return lambda a : a * n
```

```
mydoubler = myfunc(2)

print(mydoubler(11))
```

VIII. <u>Python Classes and Objects</u>

Python is an oop. A class is like blueprint for creating objects.

1) How to create a Class?

To create a class, use the keyword class:

Eg:

Create a class named MyClass, with a property named x:

```
Class MyClass:
    x= 5
```

2) How to create an object?

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

3) __init__() Function

All classes have a function called __init__(), which is always executed when the class is being initiated. Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Eg:

```
class Person:
    def __init__(self, name,age):
            self.name = name

            self.age = age

    p1 = Person("John",36)

    print(p1.name)

    print(p1.age)
```

4) Object Methods:

Objects can also contain methods. Methods in objects are functions that belong to the object.

Eg:

Insert a function that prints a greeting and execute it on the p1 object

```
class Person:

    def __init__(self, name,age):
            self.name = name
            self.age = age

        def display(self):
            txt = "{} is {} years old"
            print(txt.format(self.name, self.age))

 p1 = Person("John",36)
 p1.display()
```

5) **Self Parameter:** The self parameter is a reference to the current instance of the class and is used to access variables that belongs to the class. It does not have to be named self, you can call it whatever you like, but it has to be the first parameter of any function in the class:

   Eg:

```
class Person:
    def __init__(self, name, age):
            self.name = name
            self.age = age
    def display(self):
            txt = "{} is {} years old"
            print(txt.format(self.name, self.age))

            p1 = Person("John",36)
            p1.display()
```

6) **Delete Object Properties**
   You can delete properties on objects by using the del keyword:
   Eg:
   del p1.age

7) **Delete Objects**

   You can delete objects by using the del keyword

   Eg:

   del p1

IX. Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class. Parent class is the class being inherited from, also called base class. Child class is the class that inherits from another class, also called derived class.

1) **Create a Parent class:**
   Create a Parent class Person, with firstname and lastname properties, and printname method:

```
Class Person:
     def __init__(self, fname,lname):
               self.fname = fname
               self.lname = lname
     def printname(self):
               print(self.firstname,self.lastname)

x = Person("John", "Doe")
x.printname()
```

2) **Create a Child class:** To create a child class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:
   Eg:

```
Class Student(Person):
     pass

x = Student("Mike","Olsen")
x.printname()
```

3) **Add the __init__() function**
   We want to add the __init__() function to the child class
   Eg:

```
class Student(Person):
  def __init__(self, fname, lname):
```

   Eg:

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

4) **super() function:** Python also has a super() function that will make the child class inherit all the methods and properties from its parent:

```
class Student(Person):
     def __init__(self,fname,lname):
               super().__init__(fname,lname)
```

5) **Add Properties:** Add a property called graduationyear to the Student class:
   Eg:

```
class Student(Person):
    def __init__(self,fname,lname):
            super().__init__(fname,lname)
            self.graduationyear = 2019
```

Also the above statement can also be written as

Eg:

```
class Student(Person):
    def __init__(self,fname,lname,year):
            super().__init__(fname,lname)
            self.graduationyear = year
```

6) **Add Methods:** Add a method to the above code

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of",
self.graduationyear)
```

X. Python Scope

A variable is only available from inside the region it is created. This is called scope.

1) **Local Scope**: A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.

Eg:

```
def myfunc():
    x = 300
    print(x)
myfunc()
```

2) **Global Scope:** A variable created in the main body of the Python code is a global variable and belongs to the global scope. Global variables are available from within any scope, global and local.

Eg:

```
x = 300
def myfunc():
    print(x)

myfunc()

print(x)
```

Also, use the global keyword if you want to make a change to a global variable inside a function. To change the value of a global variable inside a function, refer to the variable by using the global keyword:

Eg:

```
x = 300

def myfunc():
  global x
  x = 200

myfunc()

print(x)
```

XI. **Python Iterators:**

An iterator is an object that contains a countable number of values.
An iterator is an object that can be iterated upon, meaning that you can traverse through all the values. Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods __iter__() and __next__().

Iterator vs Iterable: Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable containers which you can get an iterator from. All these objects have a iter() method which is used to get an iterator:

Eg:
```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

Output:
```
apple
banana
cherry
```

Even strings are iterable objects, and can return an iterator:
Eg:
```
mystr = "banana"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

Eg:
```
b
a
```

```
n
a
n
a
```

1) Lopping through an iterator: We can also use a for loop to iterate through an iterable object:
   Eg:
   ```
   mytuple = ("apple", "banana", "cherry")

   for x in mytuple:
     print(x)
   ```

XII. <u>Python File Handling:</u>

File handling is important for web applications.

The key function for working with files in python is the open() function. The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:
"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition u can specify if the file should be handled as binary or text mode
"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Syntax:
```
f = open("demofile.txt")
```

1) <u>To open a file in read mode and read the contents of the file</u>

```
f = open("text.txt","r")

print(f.read())
```

2) <u>To read only a specified number of characters in the file</u>

Eg:
```
f = open("text.txt","r")
```

```
print(f.read(11))
```

3) <u>**To return one line of sentence we can use readline() method**</u>

Eg:

```
f = open("text.txt","r")

print(f.readline())
```

4) <u>**By looping through the lines of the file, you can read the whole file line by line:**</u>

Eg:

```
f = open("text.txt","r")

for i in f:
        print(i)
```

5) <u>**To close a file, we make use of close method**</u>

   f.close()

6) <u>**Write to an Existing File**</u>

   To write to an existing file, you must add a parameter to the open() function:

   "a" = Append – will append to the end of the file
   "w"= Write – will overwrite any existing content

   Eg:

```
f = open("text.txt","a")

f.write("This will change the content of the file")

f.close()
```

   Eg:

```
f = open("text.txt","w")

f.write("This will change the content of the file")

f.close()
```

7) <u>**Create a New File:**</u>

   To create a new file in Python use open() method, with one of the following parameters:
   "x" – Create – will create a file, returns an error if the file exists
   "a" – Append – will create a file if the specified file does not exist
   "w" - Write – will create a file if the specified file does not exist

8) <u>**Delete File:**</u>

   To delete a file, you must import the OS module, and run its os.remove() function:

```
import os
os.remove(“demofile.txt”)
```

9) **To Check if the file exists:**

To avoid getting error, check if the file exists before u try to delete it:

Eg:

```
Import os
if  os.path.exists(“demofile.txt”):
     os.remove(“demofile.txt”)
else:
     print(“The file does not exist”)
```

10) **Delete folder**

To delete an entire folder, use to os.rmdir method:

Eg:

```
import os
os.rmdir(“myfolder”)
```

XIII. Python Modules

1) How to create a module

To create a module just save the code you want in a file with file extension .py

Eg:

Save this code in addition.py

```
def add(a,b):
        return a+b
```

Save this code in python.py

```
import addition
print(addition.add(2,3))
```

Output: 5

Eg2:

Save this file as person.py

```
person={
        "name" : "Jane Doe",
        "age":25,
        "salary":2500
}
```

Save this file as name.py

```
import person
a = person.person["name"]
print(a)
```

2) Importing existing modules

```
import platform
x=platform.system()
print(x)
```

3) To list all the functions, present in a module use dir

Eg:

```
import platform

x = dir(platform)
print(x)
```

XIV. Python Date and Time

A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

Eg:

```
import datetime
x = datetime.datetime.now()
print(x)
Output: 2022-04-11 18:06:29.515866
```

1. **Date Output:** When we execute the code from the example above the result will be 2022-04-11 18:06:29.515866. The date contains year, month, day, hour, minute, second, microsecond.

   Eg:

   Return the year and name of weekday:

   ```
   import datetime
   x = datetime.datetime.now()
   print(x.year)
   print(x.strftime("%A"))
   ```

2. **Creating Date objects:** To create a date, we can use the datetime() class(constructor) of the datetime module. The datetime() class requires three parameters to create a date: year, month, day.

   Eg:

   ```
   import datetime
   x = datetime.datetime(2020,5,17)
   print(x)
   ```

3. **The strftime() Method**

   The datetime object has a method for formatting date objects into readable strings. The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

   Eg:

   ```
   import datetime
   x = datetime.datetime(2018,6,1)
   print(x.strftime("%B"))
   ```

Format Specifiers

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365

%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%C	Century	20
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%
%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

XV. <u>Python Math</u>

The min() and max() functions can be used to find the lowest or highest value in an iterable:
Eg:

```
x = min(5,10,25
y = max(5,10,25)
print(x)
print(y)
```

The abs() function returns the absolute value of the specified number.
Eg:

```
x = abs(-7.25)
print(x)
```

The pow(x,y) function returns the value of x to the power of y.

```
x = pow(4,3)
print(x)
```

1. <u>The Math Module</u>

Python has also a built-in module called math, which extends the list of mathematical functions. To use it, you must import the math module:

The math.sqrt() method returns the square root of a number.
Eg:

```
import math
x = math.sqrt(64)
print(x)
```

The math.ceil() method rounds a number upwards to its nearest integer, and the math.floor() method rounds a number downwards to its nearest integer and returns the result:
Eg:

```
import math
x=math.ceil(1.4)
y=math.floor(1.4)
print(x) #returns 2
print(y) #returns 1
```

The math.pi constant returns the value of PI (3.14)
Eg:

```
Import math
x = math.pi
print(x)
```

XVI. <u>Python RegEx</u>

A regex is a sequence of characters that forms a search pattern. Regex can be used to check if a string contains the specified search pattern.
Regex can be used to check if a string contains the specified search pattern

1) Regex Module: Python has a built in package called re, which can be used to work with Regular Expressions.
   Import re module:
   import re

2) RegEx in Python
   Search the string to see if it starts with “The” and ends with “Spain”:

   Eg:

   ```
   import re
   txt = “The rain in Spain”
   x = re.search(“^The.*Spain$”,txt)
   ```

3) Regex Functions

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

4) Metacharacters

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurrences	"he.*o"

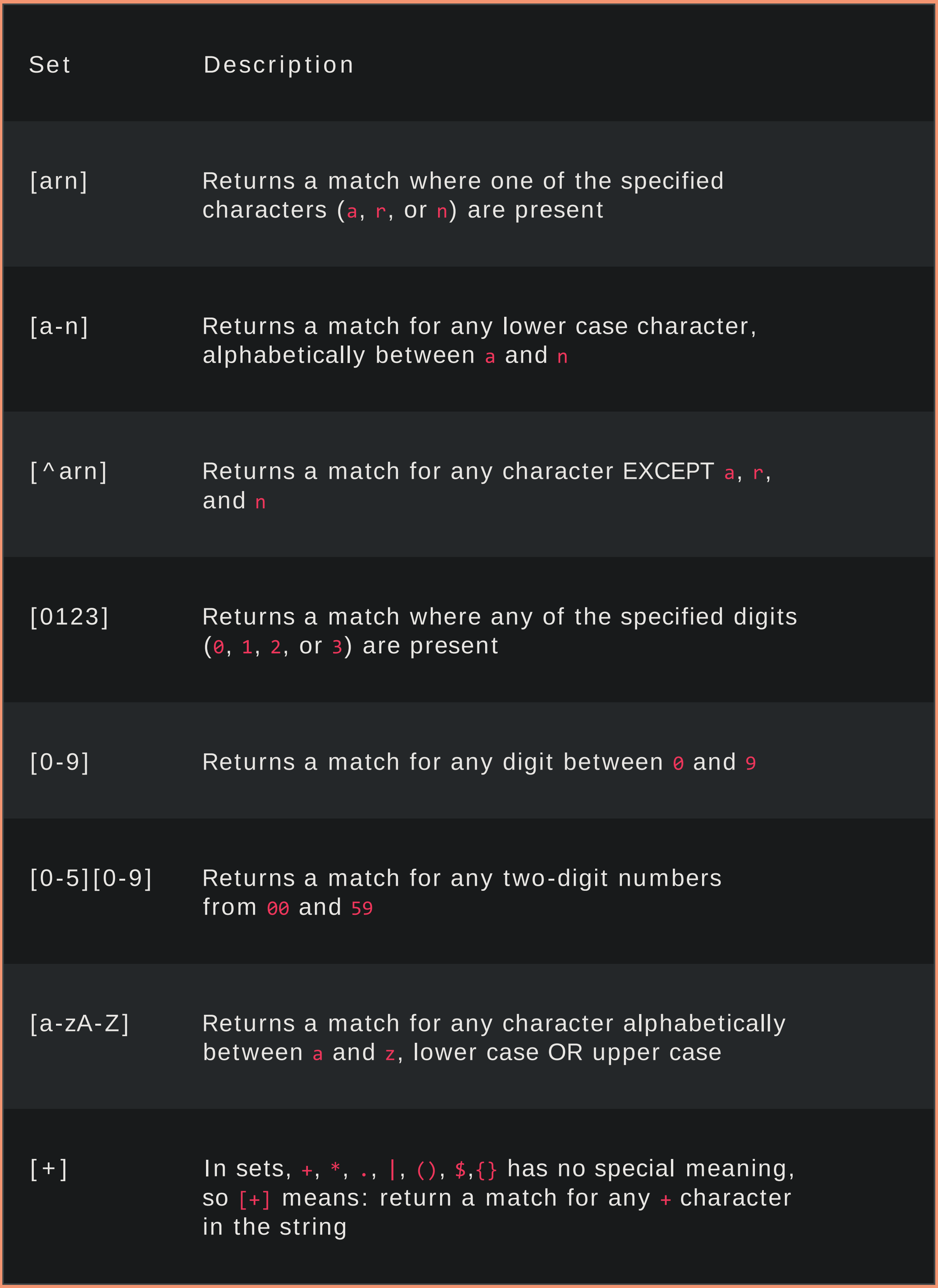
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{}	Exactly the specified number of occurrences	"he.{2}o"
|	Either or	"falls|stays"
()	Capture and group	

5) <u>Special Sequences</u>

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at	r"\Bain" r"ain\B"

	the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

6) Sets: A set is a set of characters inside a pair of square brackets [] with a special meaning:



Set	Description
[arn]	Returns a match where one of the specified characters (`a`, `r`, or `n`) are present
[a-n]	Returns a match for any lower case character, alphabetically between `a` and `n`
[^arn]	Returns a match for any character EXCEPT `a`, `r`, and `n`
[0123]	Returns a match where any of the specified digits (`0`, `1`, `2`, or `3`) are present
[0-9]	Returns a match for any digit between `0` and `9`
[0-5][0-9]	Returns a match for any two-digit numbers from `00` and `59`
[a-zA-Z]	Returns a match for any character alphabetically between `a` and `z`, lower case OR upper case
[+]	In sets, `+`, `*`, `.`, `|`, `()`, `$`,`{}` has no special meaning, so `[+]` means: return a match for any `+` character in the string

7) **<u>The findall() method:</u>** returns a list containing all matches.

Eg:

```
import re
txt = “The rain in Spain”
x = re.findall(“ai”,txt)
print(x)
```

The list contains the matches in the order they are found. If no matches are found, an empty list is returned:

Eg:

```
import re
txt = "The rain in Spain"
x = re.findall("Portugal", txt)
print(x)
```

8) The search() function

The search() function searches the string for a match, and returns a match object if there is a match. If there is no more than one match, only the first occurrence of the match will be returned.

Eg:

```
import re
txt = "The rain in Spain"
x = re.search("\s", txt)
print("The first white-space character is located in position:", x.start())
```

9) The split() function

Returns a list where the string has been split at each match:

Eg:

```
import re
txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

10) The sub() function

The sub() function replaces the matches with the text of your choice:

Eg:

```
import re
txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

count can be used to control the number of replacements

Eg:

```
import re
txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2)
print(x)
```

11) The Match object

A match object is an object containing information about the search and the results.
Eg:

```
import re
txt = "The rain in Spain"
x = re.search("ai", txt)
print(x) #this will print an object
```

The Match object has properties and methods used to retrieve information about the search, and the result:

.span() returns a tuple containing the start-, and end positions of the match.
.string returns the string passed into the function
.group() returns the part of the string where there was a match

Eg:
Print the position (start- and end-position) of the first match occurrence.
The regular expression looks for any words that starts with an upper case "S":

```
import re
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.span())
```

Eg:
Print the string passed into the function:

```
import re
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.string)
```

Eg:
Print the part of the string where there was a match.
The regular expression looks for any words that starts with an upper case "S":

```
import re
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.group())
```

XVII. <u>Python User input</u>

Python allows for user input. The following example asks for the username, and when you entered the username, it gets printed on the screen:
Eg:

```
username = input("Enter username:")
print("Username is: " + username)
```