IOTA ACADEMY
Become Job Ready...

Types Of Functions

1. Built-in Functions
2. User-defined Functions

Built-in Functions

1. abs()

```
In [21]:
```
```
# find the absolute value

num = -100

print(abs(num))
```
```
100
```

2. all()

return value of all() function

True: if all elements in an iterable are true

False: if any element in an iterable is false

```
In [22]:
```
```
lst = [1, 2, 3, 4]
print(all(lst))
```
```
True
```
```
In [23]:
```
```
lst = [0, 2, 3, 4]    # 0 present in list
print(all(lst))
```
```
False
```
```
In [24]:
```
```
lst = []              # empty list always true
print(all(lst))
```
```
True
```
```
In [25]:
```
```
lst = [False, 1, 2]   # False present in a list so all(lst) is False
print(all(lst))
```
```
False
```

dir()

The dir() tries to return a list of valid attributes of the object.

If the object has **dir()** method, the method will be called and must return the list of attributes.

If the object doesn't have **dir()** method, this method tries to find information from the **dict** attribute (if defined), and from type object. In this case, the list returned from dir() may not be complete.

```
In [26]:
```
```
numbers = [1, 2, 3]

print(dir(numbers))
```
```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

divmod()

The divmod() method takes two numbers and returns a pair of numbers (a tuple) consisting of their quotient and remainder.

```
In [27]:
```
```
print(divmod(9, 2)) # print quotient and remainder as a tuple

# try with other number

(4, 1)
```
```
In [28]:
```
```
quotient, remainder = divmod(10,3)

print("I am quotient",quotient)
print("I am remainder",remainder)
```
```
I am quotient 3
I am remainder 1
```

enumerate()

The enumerate() method adds counter to an iterable and returns it

syntax: enumerate(iterable, start=0)

```
In [29]:
```
```
numbers = [10, 20, 30, 40]

for index, num in enumerate(numbers):

    print("index {index} has value {num}")


index 0 has value 10
index 1 has value 20
index 2 has value 30
index 3 has value 40
```
```
In [30]:
```
```
list(enumerate(numbers,20))
```
```
Out[30]:
```
```
[(20, 10), (21, 20), (22, 30), (23, 40)]
```
```
In [31]:
```
```
l1 = ["eat","sun","1","is"]
l2 = ['y','ne','s','ta']

# to iterate
for i,j in enumerate(l2):
    l1.append(l1[i]+j)

print(l1)
```
```
['My', 'name', 'is', 'iota']
```
```
In [32]:
```
```
# The zip() function takes iterables, aggregates them in a tuple, and returns it.

l1 = ["eat","sun","1","is"]
l2 = ['y','ne','s','ta']

l = []
for i,j in zip(l1,l2):
    l.append(i+j)

print(l)
```
```
['My', 'name', 'is', 'iota']
```
```
In [33]:
```
```
list(zip(l1,l2))
```
```
Out[33]:
```
```
[('M', 'y'), ('na', 'me'), ('i', 's'), ('io', 'ta')]
```

filter()

The filter() method constructs an iterator from elements of an iterable for which a function returns true.

syntax: filter(function, iterable)

```
In [34]:
```
```
def even_odd(x):
    if x%2==0:
        return x
```
```
In [35]:
```
```
listnum = [133,34,2,4,6,37,34,22,39,41]
list(filter(even_odd,listnum))

# [132,34,2,4,6,None,34,22,None,None]  <---Return Values
# [T,T,T,T,T,F,T,T,F,F]
```
```
Out[35]:
```
```
[133, 34, 2, 4, 6, 34, 22]
```
```
In [36]:
```
```
def multiple(x):
    if num%4==0:
        return num
```
```
In [39]:
```
```
listnum = [132,34,2,4,6,421]
list(filter(multiple,listnum,listnum))

# [132,None,None,None,6,None]
# [T,F,F,F,T,F]
```
```
Out[39]:
```
```
[132, 6]
```
```
In [42]:
```
```
def find_positive_number(num):
    """
    This function returns the positive number if num is positive
    """
    if num > 0:
        return num
```
```
In [43]:
```
```
number_list = range(-10, 10) # create a list with numbers from -10 to 10
print(list(number_list))

# we will see this later: what if we use map here?
positive_num_list = list(filter(find_positive_number, number_list))

print(positive_num_list)
```
```
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

isinstance()

The isinstance() function checks if the object (first argument) is an instance or subclass of classinfo class (second argument).

syntax: isinstance(object, classinfo)

```
In [44]:
```
```
lst = [1, 2, 3, 4]
print(isinstance(lst, list))  # checks if the variable belongs to the specified class or not

# try with other datatypes tuple, set
t = (1,2,3,4)
print(isinstance(t, list))
```
```
True
False
```
```
In [47]:
```
```
print(isinstance.__doc__)
```
```
Return whether an object is an instance of a class or of a subclass thereof.

A tuple, as in ``isinstance(x, (A, B, ...))``, may be given as the target to
check against. This is equivalent to ``isinstance(x, A) or isinstance(x, B)
or ...`` etc.
```

map()

Map applies a function to all the items in an input_list.

syntax: map(function_to_apply, list_of_inputs)

```
In [45]:
```
```
numbers = [1, 2, 3, 4]

# normal method of computing num^2 for each element in the list.
squared = []
for num in numbers:
    squared.append(num ** 2)

print(squared)
```
```
[1, 4, 9, 16]
```
```
In [48]:
```
```
def sqr(num):
    return num**2

# using map() function
squared = list(map(sqr, numbers))
print(squared)
```
```
[1, 4, 9, 16]
```
```
In [49]:
```
```
# computing factorial for each element in the list using map function.
numbers = [1, 2, 3, 4]

def factorial(num):
    fac = 1
    for i in range(1,num+1):
        fac = fac*i
    return fac

# using map() function
facto = list(map(factorial, numbers))  # what if we apply filter here?? It will give the same output
print(facto)                           # No, filter will not give the same output
```
```
[1, 2, 6, 24]
```

reduce()

reduce() function is for performing some computation on a list and returning the result.

It applies a rolling computation to sequential pairs of values in a list.

```
In [53]:
```
```
# product of elements in a list
product = 1
lst = [1, 2, 3, 4,5]

# traditional program without reduce()
for num in lst:
    product *= num
print(product)
```
```
120
```
```
In [52]:
```
```
# with reduce()
from functools import reduce # in Python 3. It's built in for python 2, but we can import it :)

def multiply(x,y):
    return x*y

product = reduce(multiply, lst)
print(product)
```
```
120
```

2. User-defined Functions

Functions that we define ourselves to do certain specific task are referred as user-defined functions.

If we use functions written by others in the form of library, it can be termed as library functions.

Advantages

1. User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.
2. If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
3. Programmars working on large project can divide the workload by making different functions.

Example:

```
In [55]:
```
```
def product_numbers(a, b):  # this is a user-defined function
    """
    this function returns the product of two numbers
    """
    product = a * b
    return product

num1 = 10
num2 = 20

print(f"product of {num1} and {num2} is {product_numbers(num1,num2)} ")
```
```
product of 10 and 20 is 200
```

Python program to make a simple calculator that can add, subtract, multiply and division

```
In [56]:
```
```
def add(a, b):
    """
    This function adds two numbers
    """
    return a + b

def multiply(a, b):
    """
    This function multiply two numbers
    """
    return a * b

def subtract(a, b):
    """
    This function subtract two numbers
    """
    return a - b

def division(a, b):
    """
    This function divides two numbers
    """
    return a / b

print("Select Option")
print("1. Addition")
print("2. Subtraction")
print("3. Multiplication")
print("4. Division")

# take input from user
choice = int(input("Enter choice 1/2/3/4"))

num1 = float(input("Enter first number:"))
num2 = float(input("Enter second number:"))
if choice == 1:
    print(f"Addition of {num1} and {num2} is {add(num1, num2)}")
elif choice == 2:
    print(f"Subtraction of {num1} and {num2} is {subtract(num1, num2)}")
elif choice == 3:
    print(f"Multiplication of {num1} and {num2} is {multiply(num1, num2)}")
elif choice == 4:
    print(f"Division of {num1} and {num2} is {division(num1, num2)}")
else:
    print("Invalid Choice")
```
```
Select Option
1. Addition
2. Subtraction
3. Multiplication
4. Division

Addition of 10.0 and 2.0 is 12.0
```

Great Job!