Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

In [1]:
```
def my_function(fname):
  print(fname + " Refsnes")

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

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

In [2]:
```
def my_function(fname, lname):
  print(fname + " " + lname)


my_function("Emil", "Refsnes")
```

```
Emil Refsnes
```

In [3]:
```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_2148\2663467011.py in <module>
      2   print(fname + " " + lname)
      3 
----> 4 my_function("Emil")

TypeError: my_function() missing 1 required positional argument: 'lname'
```

*args

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

In [ ]:
```
def my_function(*kids):
  print("The youngest child is " + kids[-1])

my_function("Emil", "Tobias", "Linus")
```

**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.

In [4]:
```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "riya", lname = "wagh")
```

```
His last name is wagh
```

Default Parameter Value

In [5]:
```
def my_function(country = "Norway"):
  print("I am from " + country)

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

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

Recursion

- Python also accepts function recursion, which means a defined function can call itself.
- Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

In [6]:
```
def recursion(k):
    if(k > 0):
        result = k + recursion(k - 1)
        print(result)
    else:
        result = 0
    return result

print("\n\nRecursion Example Results")
recursion(6)
```

```


Recursion Example Results
1
3
6
10
15
21
```

Out[6]:
```
21
```

Python Lambda

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

Syntax

lambda argument(s) : expression

In [7]:
```
x = lambda a: a*a
print(x(5))
```

```
25
```

In [8]:
```
x = lambda a , b: a*b
print(x(5,8))
```

```
40
```

In [9]:
```
x = lambda a,b,c: a*b*c
print(x(5,8,5))
```

```
200
```

In [10]:
```
def num(n):
    return lambda a : a * n
x =num(6)
y=x(5)
print(y)
```

```
30
```

Map Function

- The map() function executes a specified function for each item in an iterable. The item is sent to the function as a parameter.

Syntax

map(function, iterables)

In [11]:
```
def myfunc(a, b):
     return a + " and " + b

x = map(myfunc, ('apple', 'banana', 'cherry'), ('orange', 'lemon', 'pineapple')
list(x)
```

Out[11]:
```
['apple and orange', 'banana and lemon', 'cherry and pineapple']
```

In [12]:
```
a=[1,2,3,4,5]
```

In [13]:
```
list(map(lambda x : x**2 , l))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_2148\2552931991.py in <module>
----> 1 list(map(lambda x : x**2 , l))

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

In [14]:
```
list(map(lambda x: x+1,l))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_2148\3388377354.py in <module>
----> 1 list(map(lambda x: x+1,l))

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

In [15]:
```
list(map(lambda x: str(x) , l))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_2148\3535641421.py in <module>
----> 1 list(map(lambda x: str(x) , l))

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

In [16]:
```
l1 = [1,2,3,4,5]
l2=[6,7,8,9,10,11]
```

In [17]:
```
list(map(lambda x,y: x+y, l1,l1))
```

Out[17]:
```
[2, 4, 6, 8, 10]
```

In [18]:
```
f=lambda x,y:x*y
list(map(f,l1,l2))
```

Out[18]:
```
[6, 14, 24, 36, 50]
```

In [19]:
```
s="pwskills"
list(map(lambda x: x.upper(),s))
```

Out[19]:
```
['P', 'W', 'S', 'K', 'I', 'L', 'L', 'S']
```

Reduce

reduce() is defined in “functools” module.

reduce() stores the intermediate result and only returns the final summation value.

reduce(fun, seq) takes functionas 1st and sequence as 2nd argument. .

- At first step, first two elements of sequence are picked and the result is obtained.
- Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.
- This process continues till no more elements are left in the container.
- The final returned result is returned and printed on console.

In [20]:
```
from functools import reduce
```

In [21]:
```
a=[1,2,3,4,5,6,7,8,9]
reduce(lambda x,y:x+y,a)
```

Out[21]:
```
45
```

In [22]:
```
a = [8,6,8,9,1,6,6,2,1,0,1]
print(a)
reduce(lambda x,y : x if x>y else y , a)
```

```
[8, 6, 8, 9, 1, 6, 6, 2, 1, 0, 1]
```

Out[22]:
```
9
```

accumulate()

- accumulate() in “itertools” module.
- accumulate() returns a iterator containing the intermediate results.
- accumulate(seq, fun) takes sequence as 1st argument and function as 2nd argument

In [23]:
```
import itertools
print(a)
list(itertools.accumulate(a,lambda x,y:x+y))
```

```
[8, 6, 8, 9, 1, 6, 6, 2, 1, 0, 1]
```

Out[23]:
```
[8, 14, 22, 31, 32, 38, 44, 46, 47, 47, 48]
```

In [24]:
```
a = [8,6,8,9,1,6,6,2,1,0,1]
print(a)
list(itertools.accumulate(a,lambda x,y : x if x>y else y ))
```

```
[8, 6, 8, 9, 1, 6, 6, 2, 1, 0, 1]
```

Out[24]:
```
[8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9]
```

abs function

- Return the absolute value of a number.
- The aggument may be an integer , a floating point number, or an object implementing **abs**().
- if the argument is a complex number, its magnitude is returned.

In [25]:
```
x = abs(-10)
print(x)
y = abs(-90.658)
print(y)
```

```
10
90.658
```

filter() method

The filter() method filter the given sequence withb the help of a function thatv tests each element in the sequence to be true or not.

syntex:

filter(function,sequence)

Paramrters:

- function that tests if each element of a sequence true or not
- sequence which need to be filtered, it can be sets, list, tuple or container of any iterators.

In [26]:
```
a = [0,1,2,3,4,5,6,7,8,9,10]
result = filter(lambda  x : x % 2 != 0 , a)
print(list(result))
result = filter(lambda  x : x % 2 == 0 , a)
print(list(result))
```

```
[1, 3, 5, 7, 9]
[0, 2, 4, 6, 8, 10]
```

In [27]:
```
def fun(x):
     if(x % 2 != 0):
            return "odd"
     else:
            return "even"
print(list(map(fun,a)))
```

```
['even', 'odd', 'even', 'odd', 'even', 'odd', 'even', 'odd', 'even', 'odd',
'even']
```