Chapter 2: Function

Prepared by: Hanan Hardan

Function

- 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.
- In Python a function is defined using the def keyword:

Function

Function not return value without Parameters:

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

To call a function, use the function name followed by parenthesis:

my_function()

Function

Function not return value with Parameters:

```
def my_function(fname):
  print(fname + " University")

my_function("Philadelphia")
my_function("Jordan")
```

Note: By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

Function

Function Return Values with Parameters:

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


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

Function

Q1: Write python program that read 2 integer numbers(x,y) then call function that return the sum of numbers from x to y

```
def sum(n,m):
    s=0
    for i in range(n,m):
       s+=i
    return s
x=int(input("Enter integer number"))
y=int(input("Enter integer number"))
print(sum(x,y))
```

Function

Q2:Write a Python function that takes a number as a parameter and check the number is prime or not.

Note : A prime number (or a prime) is a natural number greater than 1 and that has no positive divisors other than 1 and itself.

```
def test_prime(n):
    if (n==1):
        return False
    elif (n==2):
        return True;
    else:
        for x in range(2,n):
            if(n % x==0):
                return False
        return True
print(test_prime(9))
```

Function

Default Parameter Value:

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


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

Function

```
def show(name,msg='good morning'):
    print("Hello",name,msg)


show('Sami')
show(“Rami”,"How do you do")
```

Function

Keywords Arguments:

The order of the arguments can be changed by using Keywords arguments

```
def show(name,msg='good morning'):
    print("Hello",name,msg)


show("Rami",msg="How do you do")
show(msg="How do you do",name="Sami")
show(name="Rami","How do you do")
```

→ error

Function

Arbitrary Arguments:

- Used when we do not advance the number of arguments that will be passed into function

```
def show(*names):
    for n in names:
        print("Hello",n)

show("sami","Omar","Rami")
```

Function

Keyword Arguments

- You can also send arguments with the *key* = *value* syntax.
- This way the order of the arguments does not matter.

Example:

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

my_function(child1 = “Omar", child2 = “Sami", child3
= “Rami")
```

Function

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.

Example:

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

Function

Recursion

- Python also accepts function recursion, which means a defined function can call itself.
- This has the benefit of meaning that you can loop through data to reach a result.
- The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

Function

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

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

Function

In this example, tri_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

Function

Q1:Write a Python program to solve the Fibonacci sequence using recursion

- Fibonacci sequence{1,1,2,3,5,8,13,21,34,55,……}

```
def fibonacci(n):
  if n == 1 or n == 2:
    return 1
  else:
    return (fibonacci(n - 1) + (fibonacci(n - 2)))
print(fibonacci(7))
```

Function

Q2:Write a Python program to calculate the sum of the positive integers of n+(n-2)+(n-4)... (until n-x =< 0).

```
def sum_series(n):
  if n < 1:
    return 0
  else:
    return n + sum_series(n - 2)
print(sum_series(6))
print(sum_series(10))
```

Function

Q3:Write a Python program to find the greatest common divisor (gcd) of two integers.

```
def Recurgcd(a, b):
        low = min(a, b)
        high = max(a, b)
        if low == 0:
                return high
        elif low == 1:
                return 1
        else:
                return Recurgcd(low, high%low)
print(Recurgcd(12,14))
```

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 *arguments* : *expression*

- The expression is executed and the result is returned:

Python Lambda

Example 1:

Add 10 to argument a, and return the result:

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

Example 2:

Multiply argument a with argument b and return the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

Python Lambda

Example 3:

Summarize argument a, b, and c and return the result:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Python Lambda

- Why Use Lambda Functions?
- The power of lambda is better shown when you use them as an anonymous function inside another function.
- Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

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

Python Lambda

- Use that function definition to make a function that always doubles the number you send in:

```
mydoubler = myfunc(2)
print(mydoubler(11))
```

- use the same function definition to make a function that always *triples* the number you send in:

```
mytripler = myfunc(3)
print(mytriples(11))
```