Python: Functions
Mathematical functions

\[ f(x) = x^2 \]

\[ f(x,y) = x^2 + y^2 \]

In programming functions also help creating better structure with decomposition
Functions

- write reusable pieces/chunks of code, called **functions**
- functions are not run in a program until they are “called” or “invoked” in a program
- function characteristics:
  - has a **name**
  - has **parameters** (0 or more)
  - has a **docstring** (optional but recommended)
  - has a **body**
  - **returns** something

Defining and invoking a function

```python
def is_even(i):
    """Input: i, a positive int
    Returns True if i is even, otherwise False
    """
    print("inside is_even")
    return i%2 == 0

is_even(3)
```

Defining and invoking a function

```python
def is_even(i):
    """
    Input: i, a positive int
    Returns True if i is even, otherwise False
    """
    print("inside is_even")
    return i%2 == 0
```

Defining and invoking a function

Consider \( f(x) = x^2 \)

def square(x):
    #defining function
    return x*x

square(4)  #invoking function

16  # output
Defining and invoking a function

Example: Functions may not have arguments, and return statement

```python
def myprint():    #defining function
    print ("Hello world")

myprint()        #invoking function

Hello world     # output
```
Defining and invoking a function

Example: Function calling another function

```python
def repeatmyprint():
    myprint()
    myprint()

repeatmyprint()  # invoking function

Hello world     # output
Hello world
```
Scope of a Variable

def f(x):
    x = x + 1
    print('in f(x): x =', x)
return x

x = 3
z = f(x)
def f(x):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f(x)
Scope of a Variable

```python
def f(x):
    x = x + 1
    print('in f(x): x = ', x)
    return x

x = 3
z = f(x)
```

Scope of a Variable

def f(x):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f(x)

Function: Arguments

def func_a():
    print ('inside func_a')

def func_b(y):
    print ('inside func_b')
    return y

def func_c(z):
    print ('inside func_c')
    return z()

def func_a():
    print('inside func_a')
def func_b(y):
    print('inside func_b')
    return y
def func_c(z):
    print('inside func_c')
    return z()
print(func_a())
print(5 + func_b(2))
print(func_c(func_a))

returns None

Function: Arguments

def func_a():
    print('inside func_a')
def func_b(y):
    print('inside func_b')
    return y
def func_c(z):
    print('inside func_c')
    return z()
print(func_a())
print(5 + func_b(2))
print(func_c(func_a))

Function: Arguments

def func_a():
    print('inside func_a')

def func_b(y):
    print('inside func_b')
    return y

def func_c(z):
    print('inside func_c')
    return z()

print(func_a())
print(5 + func_b(2))
print(func_c(func_a))

Function: Arguments

def func_a():
    print('inside func_a')
def func_b(y):
    print('inside func_b')
    return y
def func_c(z):
    print('inside func_c')
    return z()
print(func_a())
print(5 + func_b(2))
print(func_c(func_a))

Output
inside func_a
None

Function: Arguments

def func_a():
    print('inside func_a')
def func_b(y):
    print('inside func_b')
    return y
def func_c(z):
    print('inside func_c')
    return z()
print(func_a())
print(5 + func_b(2))
print(func_c(func_a()))

Output
inside func_b
7

def func_a():
    print('inside func_a')

def func_b(y):
    print('inside func_b')
    return y

def func_c(z):
    print('inside func_c')
    return z()

print(func_a())
print(5 + func_b(2))
print(func_c(func_a))

Output:
inside func_c
inside func_a
None

Function: Scope

```python
def f(y):
    x = 1
    x += 1
    print(x)
```

`x` is redefined locally

```python
x = 5
f(x)
print(x)
```

Output

2
5

Function: Scope

```python
def g(y):
    print(x)
    print(x+1)
```

Can access `x` defined outside

```
x=5
g(x)
print(x)
```

Output

```
5
6
5
```

Function: Scope

def h(y):
    x += 1

x=5
h(x)
print(x)

Can not modify
x defined outside

Output
UnboundLocalError

Function: Scope (Example)

def g(x):
    def h():
        x = 'abc'
        x = x + 1
        print('g: x =', x)
        h()
    return x

x = 3
z = g(x)
print(z)