

ICS-104

Chapter 5

Notes

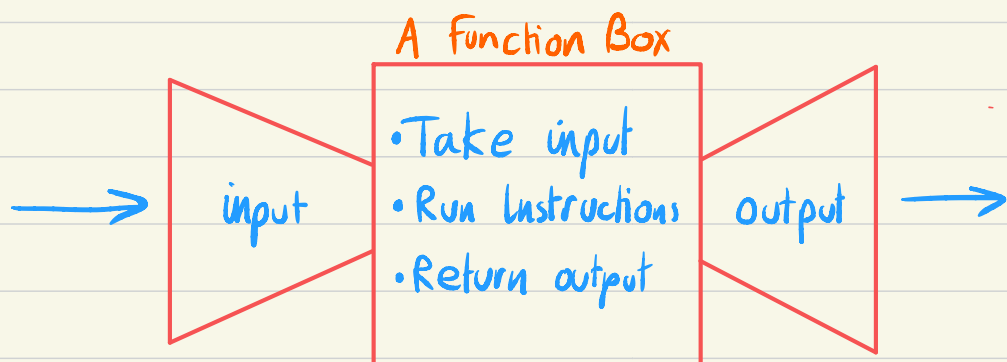
By: Naif Alqahtani

 Twitter: @NaifAlqahtani

 Youtube.com/MWNLLT

• Functions

- **Functions** are a set of instructions that can be named.
- A function can execute its instructions when its name is **called**.
- Here is a diagram of a function:



- Functions can take **inputs** like variables.
- Recall that variables passed into a function are called **Arguments**
- Inputs can be used inside the instructions of a function.
- Functions can return value. These values are called **return values**.
- Functions can take **zero** or **more arguments** (i.e. inputs) but can only **return ONE value** at **most**.
- Functions can be used to pack a set of instructions that are used frequently to avoid repetition.

Implementing Functions

- In python, to create a function, you need to write this header:

must write ↗

```
def nameOfFunction(arg1, arg2):
```

this will be the name you use to call the function later.

Don't forget the colon!

write as many formal parameters as you will need.

All code that is indented here is part of the function.

```
    # code here
```

- You can call this function anywhere in your code like this.

name you used

```
nameOfFunction(arg1, arg2)
```

Pass the variables you want the function to use

*You must pass the same number of arguments as defined earlier.

- Lets write an example.

- lets write a function that computes the area of the triangle
- $\text{Area} = \frac{1}{2} \times \text{Base} \times \text{height}$.
- From the equation, its clear we need 2 variables
 - Base
 - Height.
- We can make those 2 variable our arguments for the function

function {

```

def areaOfTriangle(base, height):
    area = 0.5 * base * height
    return area

base = 12
height = 6
area = areaOfTriangle(base, height)

print(area)
    
```

name →

function takes 2 parameter variables Base and height.

stores a variable

Return the variable one

takes values that are passed into the function.

store base value

store height value

calling function.

giving the function our variables that we declared.

the function takes the values, computes, then returns the area.

we store the return value in a variable

print return value that is stored in this variable.

- Now we made a function that contains instructions that return the area of a triangle
- We can call this function to find the area of many triangles without writing the equation everytime.

• Important Notes

- Python reads the program from top to bottom.
- This means you **CANNOT** call a function first, then declare the function later. Doing so will result in an **error!**
- Instructions inside the function will **Never** run **unless** the function is called.
- To organize code, we can set a main function.
- The insides of the main function will not run until main is called.
- We call main at the end.



`def main():` ← ① Python reads, but does NOT execute

`base = 12`

`height = 6`

`area = areaOfTriangle(base, height)`

`print(area)` ← will not run, because main is not called.

`def areaOfTriangle(base, height):` ← ② Python reads, but does not execute.
`area = 0.5 * base * height`
`return area`

← Python reads AND executes.
`main()` ③ At this point Python has seen the entire program. So all functions have been declared.

- Return statement **Stop** the function.
 - any instruction after **return** will **NOT** run.
- The function is finished when **return** is mentioned.
- Function can sometime produce output without returning any value.
- **variable scope** means the area of the program where the variable can be accessed
- All variables declared **INSIDE** any function can only be accessed **INSIDE** that function **ONLY**.
- Calling a variable **OUTSIDE** its function will cause an **error!**
- Variables defined inside functions are called **Local Variables**.
- Since **local variables** are only accessible inside each function only. We can use the same name in different functions without errors.
- Any variable that is declared outside all functions is called **global variables**.
- To use a global variable inside functions you must use **global** to use the same variable

```

>>> x = 20
>>> def f():
...     global x
...     x = 40
...     print(x)
...

>>> f()
40
>>> x
40
    
```

← calling global variable "x"

← Not using **global** means that "x" will be a local variable.

- Global variables are not a good idea because you can lose track of where it is being updated. Try to avoid using them.

Good luck!

