UCT Department of Computer Science
Computer Science 1017F

Functions

Lighton Phiri
<lphiri@cs.uct.ac.za>

April 2015

Quick reminder

- About me
  - I am Phiri
  - Email: lphiri@cs.uct.ac.za
  - I am based in the Centre for ICT4D—Computer Science Building 3A
- Block structure:
  - Functions& Testing
  - UCT CS book—Object Oriented Programming in Python 1
    - Function==Chapter 8; Testing==11.4
  - 'Prescribed' book—John Zelle
    - Function==Chapter 6

Introduction

- Unstructured sequence of statements
  - 'Some' structure—loops
- Write a program to print out the maximum of two numbers.

- For example:
  - Using 5 and 9 prints out 9

- How would we do this?
- We can use functions to make such programs reusable.

Function

- A function is a named sequence of statements that performs a specific task, and can be executed/called within a program.
- We have already used some functions:
  - `print, len, sqrt, input...`
- Python stops what it is doing, runs the function, then continues from where it stopped.

- Functions enable reuse and modularity of code.
- Functions help us to write longer/more complex programs.

Function Definition / Use

- Functions can be defined and used in any order, as long as they are used after definition.

- To define a function:

```
def function_name (...):
    “""function_name docstring“""
    statement1
    ...
```

- To use/call/invoke a function:

```
function_name ()
```

STOP 1: Function Definition / Use

- A simple exercise.
  - Write a function that adds two numbers and outputs
    - “Result”, “is”, <result of adding two numbers>

- Steps
  - Use appropriate function name
  - Use correct syntax

- To use/call/invoke the function:

```
function_name ()
```

Code reuse& duplication

- Functions can refactor code to avoid duplication

```
print (“Result”)
print (“is”)
print (1+1)
print (“Result”)
print (“is”)
print (2+2)
print (“Results”)
print (“is”)
print (3+3)
```

```
def add_fxn():
   print ("Result")
   print ("is")
add_fxn ()
print (1+1)
add_fxn ()
print (2+2)
add_fxn ()
print (3+3)
```

Input Parameters

- A function can have a list of parameters in its definition.
  - called the **formal parameters**
- Function name and corresponding parameters collectively form the
  - **function signatre**
- Whenever the function is called/invoked a value must be provided for each of the formal parameters
  - called **arguments or actual parameters**
- Within the function body, the parameters can be used like variables.

Input Parameters

- Parameters allow variation in function behaviour

```
print ("1+1=")
print (1+1)
print ("2+2=")
print (2+2)
print ("3+3=")
print (3+3)
:
:
```

```
def add_fxn(a, b):
    """add_fxn"""
    print("%s+%s=" %(a,b))
    print(a+b)


add_fxn(1, 1)
add_fxn(2, 2)
add_fxn(3, 3)
```

Input Parameters and Arguments

formal parameters

```
def add_fxn (a, b):
    print ("%s + %s = " %(a, b))
    print (a+b)

add_fxn (1, 100)
```

arguments/
actual paramters

STOP 2: Input Parameters

- A simple exercise.
  - Write a function that is able to print out the last name of any CSC1017F student
    - "<STUDENT> is enrolled for CSC1017F"

- Steps
  - Use appropriate function name
  - Use correct syntax

- How many parameters? What is function's signature?

Default Parameters

- There are times when it is desirable to have default parameters.
  - Optional parameters
- There are rules
  - All **optional parameters come after required paramters**

- Syntax

```
def function_name (first="John", last="Doe"):
   """function_name docstring"""
   Print (first," ", last)
   ...
```

STOP 3: Default Parameters

- A simple exercise.
  - Write a function that is able to print a maximum of three names—first name, maiden name (IF PERSON MARRIED)& last name
    - “<FIRST> <MAIDEN> <LAST> if married”
    - “<FIRST> <LAST> if NOT married”

- Tasks
  - How many parameters; what is function's signature; which parameters are optional?