UCT Department of Computer Science
Computer Science 1017F

Functions

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

April 2015

Functions in pespective

- Function definition
  - Function signature
- Function invocation
- Input parameters
  - Formal parameters
  - Arguments/actual parameters
- Default parameters
  - Parameter order
  - Arguments order
  - Invocation patterns

Variable Parameters - *args 1/2

- Dynamic parameter list of positional parameters.
  - * placed before formal parameter to signal use of variable turple
  - By convention, 'args' name is used
- To define a function:

```
def function_name (*args):
    """function_name docstring"""
    ...
```

- To use/call/invoke a function:

```
function_name (value, value, …)
```

Variable Parameters - *args 2/2

- An example
  - A function that prints out formal parameters a line at a time
- Steps
  - Define appropriate function signature
  - Come up with correct logic—avoid logical errors!

- Use/call/invoke the function:

```
function_name (value, value, …)
```

STOP 1: *args 1/2

- A simple exercise.
  - Write a function that takes in a variable list of numbers and prints the result of adding numbers
    - We should be able to add
      - Two, three or even 1000 numbers

- Steps
  - Define function signature—include *args
  - Use correct syntax and logic

STOP 1: *args 2/2

- A simple exercise.
  - Which of the predefined functions used thus far seem to comform to this?
    - str.upper(...)?
    - math.sqrt(...)?
    - input(...)?
    - print(...)?
    - str.count(...)

Variable Parameters - **kwargs

- Dynamic parameter list of positional parameters.
  - ** placed before formal parameter to signal use of variable dictionary
  - By convention, 'args' name is used
- To define a function:

```
def function_name (**kwargs):
    """function_name docstring"""
    ...
```

- To use/call/invoke a function:

```
function_name (key=value, key=value, ...)
```

STOP 2: **kwargs

- A simple exercise.
  - Write a function that takes in a UCT study programme name and course names associated with it
    - e.g. First year Engineering programme

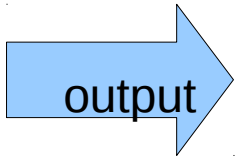
- Steps
  - Define function signature—include **kwargs
  - Use correct syntax and logic

Return Values 1/2

- Functions can return values just like mathematical functions.
- Use the **return** statement with an expression.
- Can be used anywhere in function and will return immediately.

```
def add_fxn (a, b):
    return a + b

y = add_fxn (1, 1)
print (y)
```




```
2
```

Return values 2/2

- Anything that comes after the return statement is ignored by the interpreter
- Return results can be used on right side of assignment operator in expressions
  - `y = add_fxn(2, 5)`
- Functions can have ONLY a single return value
  - Use data structure such as lists, tuples if more values returne
  - Comma seperated listing equally an option

STOP 3: Return values

- A simple exercise
  - Write a function that returns the sum of adding dynamic number of values
  - Write a function that returns the average of dynamic number of values

- Steps
  - Define function signature
  - Use correct syntax and logic

Elements of a Function

```
def add_fxn (param1, param2):
  """function_name docstring"""
  sum = param1 + param2
  return sum
```

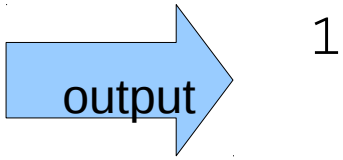
- Function signature
  - ?
- Function body
  - ?

Scope and Local Variables 1/2

- New variables can be created and used within functions but they disappear when the function ends.
  - called **local variables**

```
def print_fxn ():
    a = 1
    print (a)

Print_fxn ()
```

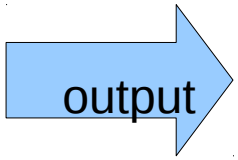


Scope and Local Variables 2/2

- Local variables names (and parameters) that are the same as global variable names temporarily hide the global variables.

```
def print_fxn (a,c):
    a = 3
    b = 3
    print (a,b)

a = 1
b = 2
print_fxn (1,2)
print (a,b)
```




```
3 3
1 2
```

Global Variables

- Global variables can be accessed but not changed.
- Use the **global** statement to allow changes to a global variable.

```
def a_fxn (a):
    global b
    b = 4
    a = 3

b = 2
a_fxn (b)
print (b)
```

output

```
4
```