Functions

Introduction to Programming

Rae Harbird, r.harbird@ucl.ac.uk

Overview

- Functions as black boxes
- Flow of control
- Parameters and return values
- Optional Parameters
- Variable scope
- Questions

Functions as Black Boxes

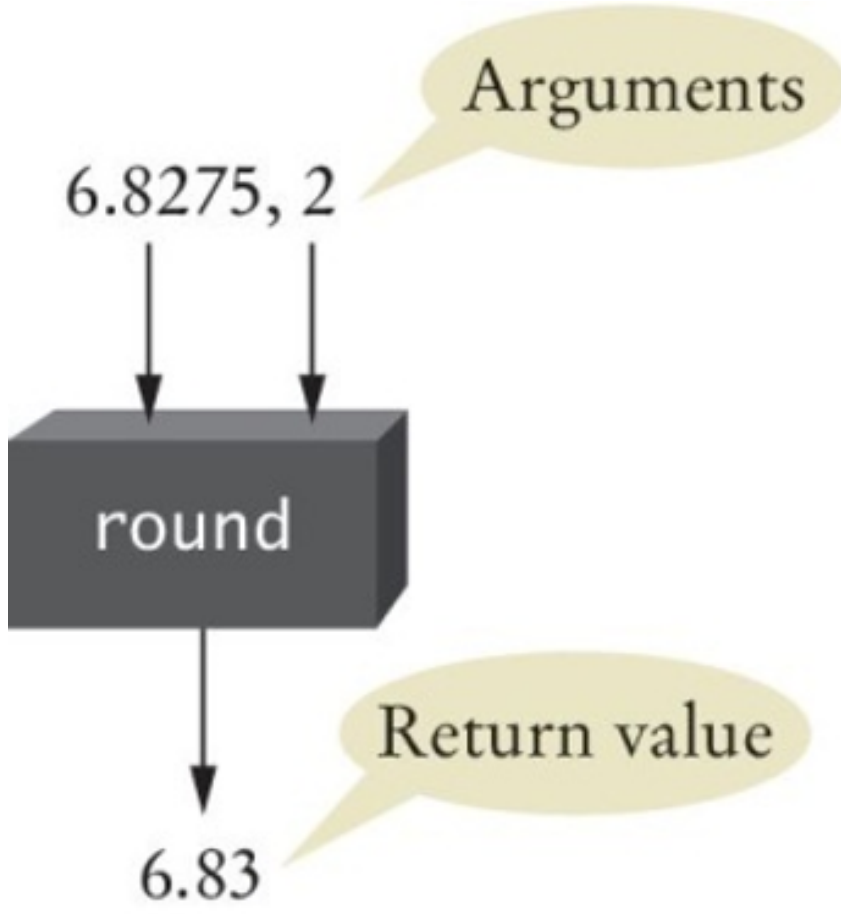

Figure 1: Function as a black box

Image: Chapter 5, Python for Everyone, 3rd Ed.

Using Functions

- Python has built-in functions and libraries containing functions
- Pass the function what it needs to do its job
- Use the result

```
# Sets cup_volume to 7.94
cup_volume = round(7.93627, 2)

# Get the user's age
age = int(input("Enter your age: "))
```

Flow of control (1/2)

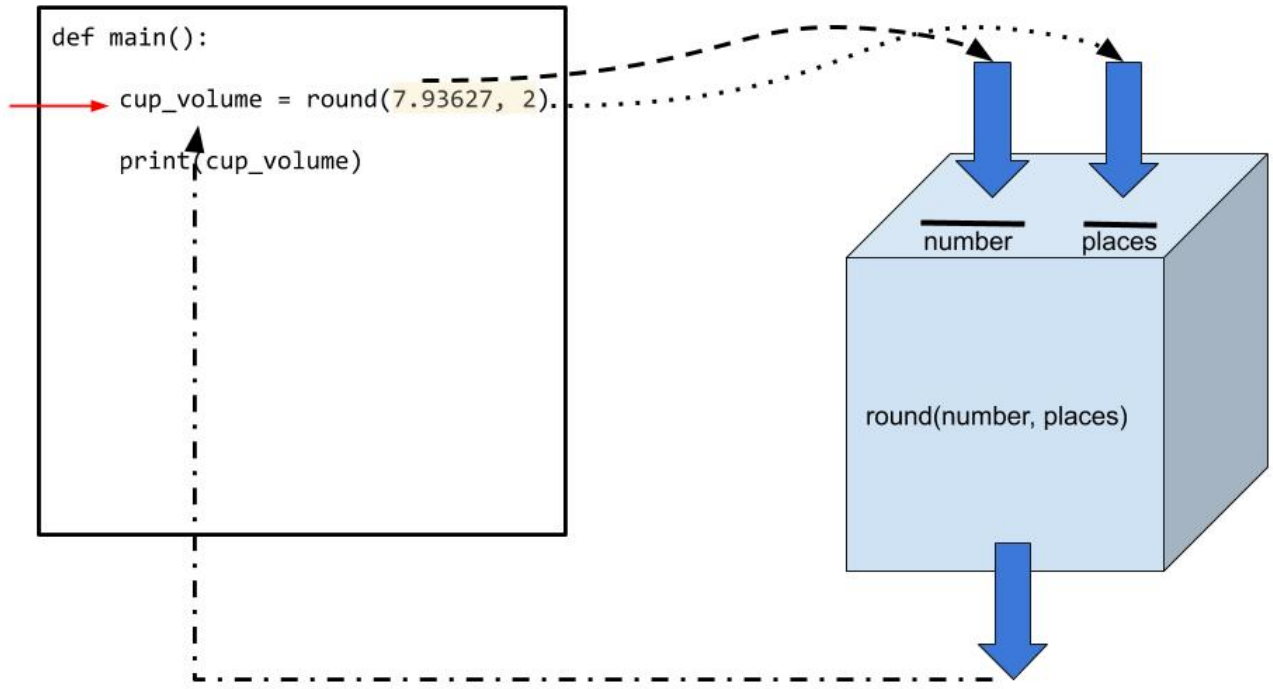

Figure 2: Flow of control (1/2)

Flow of control (2/2)

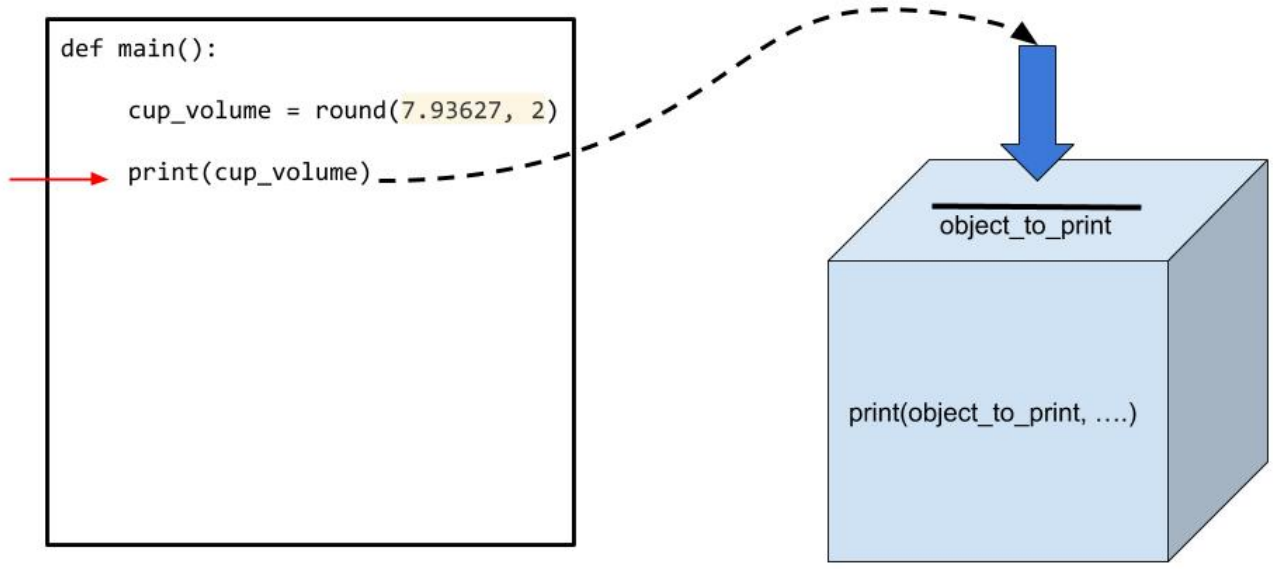

Figure 3: Flow of control (2/2)

Example With a Function

You can write ... and use, your own.

Try the version of the retirement age program which uses a function: `src/retire_with_functions.py`

Parameter Passing

Try the version of the program `guess_two_numbers.py` with no functions: `src/guess_two_numbers.py`

Now look at the version with a function that has parameters called `guess_two_numbers_again.py`.

```
def guesses_ok(answer_A, answer_B, guess_A, guess_B) :
    ok_same_order =  guess_A == answer_A and guess_B == answer_B
    ok_other_order = guess_A == answer_B and guess_B == answer_A
    return ok_same_order or ok_other_order

    ...

    if guesses_ok(first_answer, second_answer, first_guess,\
                                                second_guess):
        print("\tCorrect - well done!\n")
    else:
        print(f"\tNo, the answers were {first_answer} and \
              {second_answer}.\n")
```

Notes

- Parameter variables receive the argument values supplied in the function call
- The argument value may be:
  - The contents of a variable
  - A 'literal': `"catch"`, 22
- The parameter variable is:
  - Declared in the called function
  - Initialised with the value of the argument value
  - Used as a variable inside the called function

Parameters can be optional

Python allows you to have parameters which are optional. We have used the optional `end` parameter with `print()`

```
print("cat says meow, ", end="")
print("dog says bow wow")
```

Draw a box

We can do the same with our own functions. Here is a function to draw a box.

```
# Prints a rectangular box.
def draw_box(width, height):
    print("*" * width)
    for line in range(height - 2):
        print("*", "." * (width - 2), "*", sep="")
    print("*" * width)
```

There is one way, and only one way, to call this function.

```
draw_box(5,10)  # correct, 2 parameters
draw_box()      # wrong!!!
draw_box(5)     # also wrong!!!
```

Using optional parameters

```
# Prints a rectangular box.
def draw_box(width = 5, height = 5):
    print("*" * width)
    for line in range(height - 2):
        print("*", "." * (width - 2), "*", sep="")
    print("*" * width)


def main():
    draw_box(6,10)
```

There are 3 ways to call this function.

```
draw_box(5,10)  # correct, 2 parameters
draw_box()      # correct, both default values used
draw_box(6)     # correct, width will be 6,
                # default value used for height
```

Common Mistakes I

Parameter variables are local to the function, **do not try to change them**.

See: `bad_twice_number_with_functions.py`

```
def twice_number(the_number) :
    the_number = the_number * 2

def main():
    the_number = int(input("Enter a value for 'number'"))
    twice_number(the_number)
    print(f"The value of 'twice_number' is: {the_number}.")


if __name__ == "__main__":
    main()
```

Common Mistakes II

If your function returns a value, make sure it **always** returns a value for every branch of your code.

```
# Yes, please!!!
def cube_volume(side_length) :
    if side_length > 0
        return side_length ** 3
    else :
        return 0

# No, No, No
def cube_volume(side_length) :
    if side_length > 0 :
        return side_length ** 3
    # Error-no return value if side_length <= 0
```

Scope

A function may contain local variables. Local parameters are local variables, for example.

A variable can be 'seen' inside the block in which it is declared. The scope of a variable is the part of the program in which it is visible, this means that you can have variables in different functions with the same name. Think of towns which have a Main Street.

(left) © jchamp/iStockphoto; (middle) © StevenCarrieJohnson/iStockphoto; (right) © jsmith/iStockphoto.

Figure 4: Signpost

Image: Chapter 5, Python for Everyone, 2nd Ed.

The main() function

You don't need a `main()` function to write a Python program that works so why do we use `main()`?.

It is good programming practice to place all statements into functions, and to specify one function as the starting point. By convention, execution starts with the `main()` function. The final statement is required to call `main()`:

```
def twice_number(the_number) :
    the_number = the_number * 2

def main():
    the_number = int(input("Enter a value for 'number': "))
    print(f"The value of 'number' is {the_number}."
    twice_number(the_number)
    print(f"The value of 'twice_number' is: {the_number}.")

main()
```

if __name__ == "__main__":

In short:

- this way of structuring your program is good style, use it.

Longer:

- __name__ is a special variable which is set when you run a program.
- If you run the program from the command line, the value of __name__ is "__main__" and the `main()` function is run.
- Otherwise, it is assumed that the program is being imported into another program and we do not want to call `main()` because that program will invoke the functions as needed.
- Further detail can be found at the Runestone Interactive project

Python modules

Python offers us a great many useful libraries, each library is called **a module**.

The Python standard library is included in the Python distribution.

Some modules are 'built-in' and we go ahead and use the functions, examples are: `print()`, `abs()`, `round()`, `pow()`.

The others are not, we have to tell the interpreter which we want to use.

Using a module

Example: random library

```
import random

lucky_number = random.randint()
```

Example: math library

```
import math

temperature = math.ceil(21.1)
```

Modules - don't do this

Don't import a module like this:

```
from math import *

temperature = ceil(21.1)
```

As you can see, you don't need to specify the module name when using this form. This can be confusing for people reading your code, it's considered poor style.

Question 1: True or False

a. A function has exactly one return statement.
b. A function has at least one return statement.
c. A function has at most one return value.
d. A function that does not return a value never has a return statement.
e. When executing a return statement, the function exits immediately.
f. A function that does not return a value must print a result.
g. A function without parameter variables always returns the same value.

(Horstmann, p302)

Horstmann, Cay S., Rance Necaise. Python for Everyone, Interactive Edition, 2nd Edition. Wiley, 2016-05-09. VitalBook file.

Answer Q1

a. A function has exactly one return statement. **False** - a function may have 0 or more return statements.

b. A function has at least one return statement. **False** - a function may have 0 or more return statements.

c. A function has at most one return value. **True**

d. A function that does not return a value never has a return statement. **False** - a function that does not return a value can use a return statement to terminate its execution.

e. When executing a return statement, the function exits immediately. **True**

f. A function that does not return a value must print a result. **False** - a function could do anything (or nothing) else instead.

g. A function without parameter variables always returns the same value. **False** - consider, for example, a function that returns a random integer.

Question 2

Consider these functions:

```
def f(x):
    return g(x) + math.sqrt(h(x))

def g(x):
    return 4 * h(x)

def h(x):
    return x * x + k(x) - 1

def k(x) :
    return 2 * (x + 1)
```

Question 2 (cont.)

Without actually compiling and running a program, determine the results of the following function calls:

1. `x1 = f(2)`
2. `x2 = g(h(2))`
3. `x3 = k(g(2) + h(2))`
4. `x4 = f(0) + f(1) + f(2)`
5. `x5 = f(-1) + g(-1) + h(-1) + k(-1)`

Answer Q2

1. `x1 = f(2)`
2. `x2 = g(h(2))`
3. `x3 = k(g(2) + h(2))`
4. `x4 = f(0) + f(1) + f(2)`
5. `x5 = f(-1) + g(-1) + h(-1) + k(-1)`

The value of `x1` will be 39.0.

The value of `x2` will be 400.

The value of `x3` will be 92.

The value of `x4` will be 62.0.

The value of `x5` will be 0.0.