ENGG1003 - Monday Week 5

Functions

Sarah Johnson

University of Newcastle

22 March, 2021

Last compiled: March 15, 2021 10:08am Z

Functions

- A *function* is a block of code which can be *called* multiple times, from multiple places
- They are used when you want the same block of code to execute in many places throughout your code
- A function requires:
  - A name
  - (optional) A *return value*
  - (optional) One or more *arguments*

Functions in Mathematics

- In mathematics you saw functions written as:

$$y = f(x)$$

- Here, the function is called f, takes an argument of x and returns a value which is assigned to y
- Python and pure mathematics have these general ideas in common

Functions in Mathematics

- In mathematics you saw functions written as:

$$y = f(x)$$

- Here, the function is called f, takes an argument of x and returns a value which is assigned to y
- Python and pure mathematics have these general ideas in common
- The similarities stop there

Functions in Programming

- When a function is called:
  1. Program execution jumps to the function
  2. The function's code is executed
  3. Program execution jumps back to where it left off
  4. The function often 'returns' a value required by the program (functions can also return many or no values but we will assume one return value for now)
- The code inside the function can be any valid Python code, not just mathematics

Function Examples

- So far, we have already used several functions
- These are functions are either built in python functions or imported from the libraries we have been using
- Function call syntax is:

  `name(argument1,argument2, ...)`

  though not all functions take arguments
- Some examples:
  - `sqrt()  -- from numpy library`
  - `print()  -- built in function`
  - `random()  -- from random library`

Function Examples

- Example 1:

```
1 x = random.random()
```

  - `random` is the function name (from the library `random`)
  - It returns a random number between 0.0 and 1.0
  - The return value is assigned to `x`
  - It doesn't take an argument

Function Examples

- Example 1:

```
1 x = random.random()
```

  - `random` is the function name (from the library `random`)
  - It returns a random number between 0.0 and 1.0
  - The return value is assigned to `x`
  - It doesn't take an argument

- Example 2:

```
1 y = numpy.sqrt(x)
```

  - `sqrt` is the function name (from the library `numpy`)
  - `x` is the argument
  - It returns the square root of `x`
  - The return value is assigned to `y`

Function Examples

- Example 3:

```
1 print('Hello World')
```

  - `print` is the function name (it's a built-in function)
  - The string `'Hello World'` is the argument
  - There is no return value (technically it returns 'None')
  - Even though it does not return a value it still does something
  - Another example would be the `seed()` function

Return Values (an Engineer's View)

- The function's *return value* is the number a function gets "replaced with" in a line of code

Return Values (an Engineer's View)

- The function's *return value* is the number a function gets "replaced with" in a line of code
- Function return values, variables, and literals can all be used in the same places:
  - In arithmetic
  - In conditions
  - As arguments to other functions

Return Values (an Engineer's View)

- The function's *return value* is the number a function gets "replaced with" in a line of code
- Function return values, variables, and literals can all be used in the same places:
  - In arithmetic
  - In conditions
  - As arguments to other functions
- The following are all valid:
  - `x = rand()`
  - `print("Sine {} is {}".value(x,sin(x)))`
  - `if( (rand()%6) < 2)`
  - `while( cos(x) < 0 )`

Writing Functions

Writing your own functions:

1. Decide (for yourself) what the function needs to do

Writing Functions

Writing your own functions:

1. Decide (for yourself) what the function needs to do
2. Choose a name

Writing Functions

Writing your own functions:

1. Decide (for yourself) what the function needs to do
2. Choose a name
3. Decide on the function parameters

Writing Functions

Writing your own functions:

1. Decide (for yourself) what the function needs to do
2. Choose a name
3. Decide on the function parameters
4. Decide on the return value(s)

Writing Functions

Writing your own functions:

1. Decide (for yourself) what the function needs to do
2. Choose a name
3. Decide on the function parameters
4. Decide on the return value(s)
5. Define the function (write the function code)
6. Call the function in your code where you need to use it

Writing Functions - Example

- Lets see our code to calculate the height of a ball (from week 1) as a function

```
1 # Function Definition
2 def ball_height(v0, t):                    # Function header
3     g = 9.81                               # Function body
4     y = v0*t - 0.5*g*t**2
5     return y                               # Return statement
```

Writing Functions - Example

- Lets see our code to calculate the height of a ball (from week 1) as a function

```
# Function Definition
def ball_height(v0, t):                    # Function header
    g = 9.81                               # Function body
    y = v0*t - 0.5*g*t**2
    return y                               # Return statement
```

- Here is how we could use this function in our main program

```
v0 = 5
time1 = 0.6
height1 = ball_height(v0,time1)
time2 = 0.9
height2 = ball_height(v0,time2)
```

Function Definition

The function definition must be *before* the function's first use

- This could be in a library which must be *included* before the function is called
- Or in the same .py file placed before (above) the function call

Function Definition

- The first line in the function definition is the function *header*

```
1 def ball_height(v0, t):
2
```

- The header starts with the reserved word `def`
- Followed by the function name (here `ball_height`) and
- Followed by the function *parameters* in brackets (here the initial velocity and the time point to calculate the height)
- Ending with a colon

Function Definition

- After the function header is the function body which is all the lines of code inside the function
- The block of statements inside the function body must be indented
- An optional docstring describing the function can be added as the first line in the function body

```
1       def ball_height(v0, t):
2        """ Calculates the height of a ball at time t
3            given an initial velocity of v0 """
4        g = 9.81
5        y = v0*t - 0.5*g*t**2
6        return y
```

`return` Statements

- The function will jump back when it hits a `return` statement or the end of the function's code
- Functions which return a value *must* have a `return` statement
- Functions which return nothing don't need one
- Omitting it will cause the function to return `None`
- Examples:

```
1          return
2          return x
3          return v0*t - 0.5*g*t**2
4          return 1
5
```

Function Example

... Do it live!

Writing Functions

- Let's view a common error

```
Traceback (most recent call last):
File "Path/main.py", line 14, in <module>
y = array_sum(array_x)
NameError: name 'array_sum' is not defined
```

- The function definition is missing. Some possible causes:
  - The function definition may be below the first call on line 14
  - You may have forgotten to include the library which defines it
  - The function exits but you have a typo in the name

Writing Functions - Terminology Reminder

```
1  # Function Definition
2  # Define by function_name(function parameters)
3  def ball_height(v0, t):                    # Function header
4      """ Details .... """                   # Docustring
5      g = 9.81                               # Function body
6      y = v0*t - 0.5*g*t**2                  # Function body
7      return y                               # Return statement
8
9  # Main Program
10 # Function call via function_name(function arguments)
11 v0 = 5
12 time1 = 0.6
13 height1 = ball_height(v0,time1)  # Function call
14 time2 = 0.9
15 height2 = ball_height(v0,time2)  # Function call
```

Function Variables

- The input variables specified in the function definition are called the *parameters* of the function. The function the values provided in the call are called *arguments*
- Variable names of the arguments in function calls do not have to have the same name as in the parameters in the function definition

Function Variables

- By default, function arguments are "passed by value"
- The function gets *copies* of the variable's value
- Modifying them in a function doesn't change the original variable
  - No, not even if they have the same name
  - The function's copy occupies a different memory address

Function Variables

- Any *local variables*, those defined inside the function, are only known inside the function (e.g. `g` in the function above)
- The argument variables and local variables are discarded when the function finishes (returns)
- The return value is the *only thing* that goes back
- An exception is to define a variable as *global* inside the function but it's generally not recommended

Alternative Function Definitions

- So far we have seen functions with positional parameters
- *Keyword parameters* can be used to specify default values
- If both positional and keyword parameters are specified positional parameters must come first

```
1 def ball_height(v0, t, g=9.81):
2     return v0*t - 0.5*g*t**2
3
4 height = ball_height(5, 0.6)
5 more_precise_height = ball_height(5, 0.6, 9.80665)
```

Alternative Function Calls

- It is possible to use function input parameter names as *keyword arguments* in the function call
- The order of the arguments can be switched and the code can be more readable
- Keyword arguments can be used even if the function is defined with positional parameters

```
1 def ball_height(v0, t, g=9.81):
2     return v0*t - 0.5*g*t**2
3
4 # all of these function calls are correct
5 height = ball_height(v0=5, t=0.6)
6 height = ball_height(v0=5, t=0.6, g=9.80665)
7 height = ball_height(t=0.6, g=9.80665, v0=5)
```

Alternative Function Calls

- A mix or arguments can be used but all positional arguments must precede keyword arguments
- The positional arguments must be in the correct order

```
1 def ball_height(v0, t, g=9.81):
2     return v0*t - 0.5*g*t**2
3
4 # these function calls are correct
5 height = ball_height(5, 0.6, g=9.80665 )
6 height = ball_height(5, g=9.80665, t=0.6)
```

Functions With Multiple Return Values

- In the function definition return values are separated by commas
- In the function call return values are also separated by commas
- The order of the results returned from the function must be the same

```
1 import numpy as np
2
3 def circle(r):
4     return 2*np.pi*r, np.pi*r**2
5
6 circ, area = circle(5)
7 print('Circumference: {} Area: {}'.format(circ, area))
```

Writing Functions - Example 2

- Lets implement the `sqrt` algorithm from the week 3 lab as a function
- ...Then compare with `the numpy sqrt()`
- Keep it simple: fixed iteration count `n=10`

Writing Functions - Example 2

- In mathematics, calculate $\sqrt{k}$ by iterating:

$$x_n = \frac{1}{2}\left(x_{n-1} + \frac{k}{x_{n-1}}\right)$$
$$x_0 \neq 0$$

- In python we can encode this algorithm as:

```
1 # Calculate sqrt(k)
2 k = 26.0                  # Test value
3 xn = x/2.0                # Start value x0 = x/2
4 for i in range(0, 10, 1):
5   xn = 0.5*(xn + k/xn)
```

Writing Functions - Example 2

- Lets make some design decisions:
  - Name: `mySqrt()`
  - Argument: `k`
  - Return Value: `the square root of k`
- The function definition is therefore:

```
1 def mySqrt(k):
2   xn = k/2.0          # Start value x0 = x/2
3     for i in range(0, 10, 1):
4       xn = 0.5*(xn + k/xn)
5   return xn
```

Function Example

... Do it live!

More Information

- Further Reading: Section 4.1 of the course textbook
- More Practice: All the exercises in section 4.3 of the course textbook