Testing with Methods and Functions

Wrestling with Python

METHODS

Methods in Python

- We have been using other people's methods in Python for some time
  - `input` – reads something from the user
  - `print` – prints a message
  - `int` – converts a string into an integer
- These are all methods which we have been calling to do tasks for us

What is a Method?

- A method is a block of code that you can refer to by its identifier
- Whenever you refer to the method the block of code that is in the method is executed for you
- Methods can return values

Methods vs Functions

- Some Python documentation talks about functions rather than methods
- The difference between the two is very small:
  - Methods live inside objects
  - Functions can stand on their own
- You can regard the two terms as interchangeable

Why write a method?

- If you are going to use a piece of code more than once it should be in a method
  - This is because if you find faults in the code they only need to be fixed in one place
- If you want to break a problem down into sub-problems you can use a method
  - you can share work around by giving different people methods to write

Methods and Testing

- Breaking a solution down into methods also makes it easier to test
  - You can test each method individually before you add it to the whole
- This is exactly how we work in the physical world
  - The brake pads for a car can be tested before they are fitted to the car

Our first method

```
def silly():
      print("hello from silly")
```

- This method is called `silly`
- The body just prints a message
  - If you were a purist you'd call this a function, since it is not part of an object
  - But we are not purists here ☺

Using a method

```
silly()
print("...and we are back")
```

- You can call `silly` just by giving its name followed by a pair of brackets
- The program runs silly and then returns so that `"...and we are back"` is printed

What happened here?

```
bicycle()
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    bicycle()
NameError: name 'bicycle' is not defined
```

What happened here?

```
bicycle()
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    bicycle()
NameError: name 'bicycle' is not defined
```

- The method “bicycle” does not exist, and so python was unable to find and execute it
- You must define a method before you use it

RETURNING VALUES

Returning a value

- It is often useful to make a method that returns a value
- We could create one that returns a number that the user has typed in
- Python allows a method to return a value
- It uses the return keyword to do this

Returning a result

```
def FetchANumber():
    numberString = input()
    return int(numberString)
```

- The method called `FetchANumber` will fetch a number from the user and return it
- It gets the number as a string and then converts it into an integer

PARAMETERS

Parameters

- Fetching a number is all very well, but we really need a way that the method can display a prompt
- Methods can be given one or more *parameters* which are copied into the method when the method is called

Method with parameter

```
def FetchANumber(promptString):
    numberString = input(promptString)
    return int(numberString)
```

- The name of the parameter is supplied in brackets when the method is declared
- We can use the parameter value inside the method code

Using the parameter

```
def FetchANumber(promptString):
    numberString = input(promptString)
    return int(numberString)
```

- Everywhere the parameter is used the value given when the method was called is used

Default Parameters

```
def FetchANumber(promptString="Enter a number:"):
      numberString = input(promptString)
      return int(numberString)
```

- You can give a value for a parameter that will be substituted if the parameter is not supplied when the method is called
- If we don't give a prompt string the method above will use "Enter a number:"

Calling a method

```
age = FetchANumber("Enter your age: ")
```

- When the method is called the value of the parameter is passed into it
  - The thing passed into the method call is referred to as an *argument*
- This would cause the method to display `"Enter your age: "` as the prompt when it ran

Multiple Parameters

```
def sum(x,y):
    return x + y
```

- You can create a method with more than one parameter if you wish
- The above method accepts two items and adds them together

Proper Method Use

```
result = sum(2,3)
```

- This would set the value of result to 5, because that is what you get when you add the value 2 to the value 3

Adding Strings together

```
result = sum("hello ","world")
```

- This would set the value of result to "hello world" because that is what you get when you add the string "hello " to the string "world"
  - This works because the + operator in the sum method has meaning between strings and integers
  - What would happen if we had a minus method?

Methods and Madness

```
result = sum("hello ",99)
```

- What would this do?

Methods and Madness

```
result = sum(2, "hello ")
```

- What would this do?
- It would fail, since Python does not know how to add a string to an integer:

```
TypeError: unsupported operand
type(s) for +: 'int' and 'str'
```

Golden Rule

- If a method has multiple parameters you must make sure that your call to the method "makes sense"
- Otherwise the program will fail when it runs
  - Or do stupid things

Methods and Test

- As soon as you have made a method you need to think about how you would test it
- You treat the method as a "black box" that you feed values and look at the result that comes back
- The tests can be performed automatically

Testing methods

```
result = sum(0,0)
if result != 0:
    print('test failed')
else:
    print('test OK')
```

- This is a test of the `sum` method
- It proves that the method can add 0 to 0 and produce 0

Testing methods

```
result = sum(0,0)
if result != 0:
    print('test failed')
else:
    print('test OK')
```

- Does this mean that the method is now fully tested?

Testing methods

```
result = sum(0,0)
if result != 0:
    print('test failed')
else:
    print('test OK')
```

- Does this mean that the method is now fully tested?
- No. A method that just returned zero would pass this test

Test and Errors

```
def sum(x,y):
    return x * y
```

- This version of sum returns the product of the inputs, not the sum
- But it will still "work" with some inputs
  - (2,2) (0,0) would work fine

Designing Tests

- When we design tests we need to consider the kind of mistakes that programmers might make:
  - Using the wrong operators (`return x * y`)
  - Using the wrong variables (`return x + x`)

Test Warning

- Tests can only prove that a program has bugs in it
  - They can **never** prove that a program is bug free
- You perform tests to improve the quality of the code but you can never reach the point where you are sure that there are no bugs in the code

Test Warning

- Tests can only prove that a program has bugs in it
  - They can **never** prove that a program is bug free
- You perform tests to improve the quality of the code but you can never reach the point where you are sure that there are no bugs in the code – unless I wrote it ☺

Summary

- Methods allow us to break a program into a smaller chunks and reuse code
  - Methods and functions are the same thing
- A method is defined to have a particular name which is used when it is called
- A method can receive values to work on (parameters) and return a result
- Once you have made the method, you should figure out how to test it