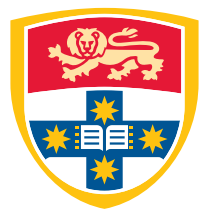
INFO1110 & COMP9001: Introduction to Programming

School of Information Technologies, University of Sydney



Copyright Warning

Lecture 16: Using objects and methods

Modules, Difference between function and a method

But first, Modules

What is a module

A single file which stores the related definitions, functions and code

Other programming languages also describe modules to aid organisation of code

Python has same goals, but also specific mechanisms associated with modules

Modules

Write the program for a supermarket checkout system

Modularity is good. Solve a subproblems in a function. Does one thing, can be easily tested. Does it well → reliable.

12 days later. 8079 lines of code.

Now we have many small functions. Too many though.

Modules (cont.)

Break up code into *developer defined categories*

e.g.
User interface subsystem
Stock tracking subsystem
Item scanning subsystem
Cash Payment subsystem
Electronic Payment subsystem
Discount subsystem

Defining a module

Done!

A module is a .py file containing source code.

HelloWorld.py is a module

HelloWorld.py

```
1 print("Hello World!")
```

Using a module

`import` is a keyword that will open another file and insert all the code *here*

ModuleTest.py

```
1 import HelloWorld
```

ModuleTest.py

```
1 import HelloWorld
2 print("what is this?")
3 import HelloWorld
```

All code is executed *once* when import is used for a module. That code is executed as if it were in the correct place.

Using a module: functions

NumberOperations.py

```
1  def get_square(x):
2      return x * x
3
4  def get_min(x, y):
5      if x < y:
6          return x
7      return y
```

ModuleTest2.py

```
1  import NumberOperations
2
3  x = 2
4  print( get_square(x) )
5  print( get_min(29, 5) )
```

No good?

Using a module: scope

Python keeps the module contents out of current scope

The current scope are all the *symbols* visible to the current execution environment

```
1  def some_func():
2      x = 2
3
4  y = 3
5  print(x)
```

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

So were are they?

Using a module: scope (cont.)

The code we have imported is stored in an Object

That object has related data and functions

There are functions and variables from an object that can be accessed using the . (dot) notation

ModuleTest2.py

```
1 import NumberOperations
2
3 x = 2
4 print( NumberOperations.get_square(x) )
5 print( NumberOperations.get_min(29, 5) )
```

Using a module: scope (cont.)

For code clarity, how about assigning that object to another variable name?

```
1 import NumberOperations as numops
2
3 x = 2
4 print( numops.get_square(x) )
5 print( numops.get_min(29, 5) )
```

Using multiple modules

Separating one python module into many has advantages for organising code

One module can import another module and this is manageable. Multiple imports can become tricky

ModuleA.py

```
1 import ModuleB
```

ModuleB.py

```
1 import ModuleC
2 def banana():
3     print("banana")
```

ModuleC.py

```
1 def callme():
2     print("This is a useful function for module A too!")
```

Using multiple modules (cont.)

For ModuleA to use a function in ModuleC, we have two ways of going about this

ModuleA.py

```
1  # go straight to the source
2  from ModuleB import ModuleC
3  ModuleC.callme()
4
5  # use a series of . operators
6  import ModuleB
7  ModuleB.banana()
8  ModuleB.ModuleC.callme()
```

This is tricky because we have to observe the hierarchy of import. It becomes even more tricky if we starting import the same module twice, in case they have code that is executed!

Methods with Objects

String type revisited

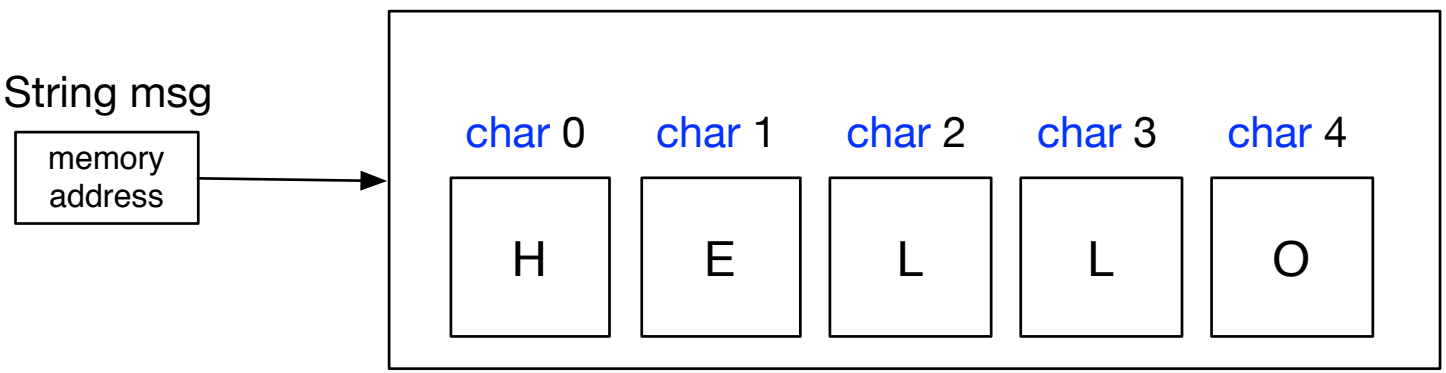
Recall the `String` type

`String` is a sequence of characters *stringed* together to represent text information. Referred to as "Text Sequence Type" in Python

`String` is an object

- it can have a few characters or several thousand.
- it has special operators to query or manipulate the text information.

```
1        msg = "HELLO"
```



Calling a function with `String` type

Write a function that will return the index of the first character in the string object that is the value `'c'`

```
1 def find_c(string):
2
3
4     .
```

Write a function that will replace the characters `'a'` with `'b'` in a string object

```
1 def replace_a_with_b(string):
2
3
4     .
```

Calling a method for `String` type

`str.find(sub[, start[, end]])` Return the lowest index in the string where substring sub is found within the slice `s[start:end]`. Optional arguments start and end are interpreted as in slice notation. Return `-1` if sub is not found.

```
1 msg = "hello world"
2 print(  msg.find('o') )
```

`str.replace(old, new))`

Return a copy of the string with all occurrences of substring old replaced by new.

```
1 msg = "hello world"
2 print(  msg.replace('o', 'a') )
```

The difference! what is it!

Function	Method
`find_value(msg, 'a')`	`msg.find('a')`
`replace_c1_with_c2(msg, 'a', 'b')`	`msg.replace('a', 'b')`

A function has no notion of existing data. It operates on inputs and produces outputs

Methods are functions operating on objects. The calling of a method will use information within the object to calculate, produce, a result

python uses functions and presents the reserved keyword `self` to refer to functions and data within the object for use as part of the function (more on this later when discussing classes).

More String methods

`str.strip()`

Return a copy of the string with the leading and trailing characters removed.

```
1 msg = "    hello world   "
2 print( msg.strip() )
3 print( msg )
```

`str.title()`

Return a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase.

```
1 String msg = "    hello world   "
2 String msg_title = msg.title()
3 print( msg )
4 print( msg_title )
```

Method call chaining

Each of these methods will return a new String.

The new String that can also have a method called on it.

```
1 msg = "     hello world     "
2 print( msg.strip().title() )
3 print( msg.title().strip()  )
4 print( msg )
```

```
1 msg = "    hello world      "
2 print(  msg.strip().replace('o','a').replace('w','k') )
```

Method call chaining (cont.)

Elaborate example of drawing a box

```
1  import sys
2
3  def print_row(width):
4      i = 0
5      while i < width:
6          print("*", end='')
7          i = i + 1
8      print("")
9
10 def print_row_stream(the_output, width):
11     i = 0
12     while i < width:
13         the_output.write("*")
14         i = i + 1
15     the_output.flush()
16
17 if __name__ == '__main__':
18     if len(sys.argv) < 2:
19         print("Please supply two integer arguments")
20         sys.exit(1)
```

Method call chaining (cont.)

```
21
22      width = int( sys.argv[1] )
23      height = int( sys.argv[2] )
24
25      if width < 1 or height < 1:
26          print("width and height should be *positive*!")
27          sys.exit(2)
28
29      print_row(width);
30      j = 0
31      while j < height - 2:
32          sys.stdout.write("*")
33          i = 0
34          while i < width - 2:
35              print(" ", end='')
36              i = i + 1
37          print("*")
38          j = j + 1
39      print_row(width);
```