Class 3: Functionalize your Programs

Lecture

Things you should do:

Why? Can reuse a function to perform the same thing, consistently!

**-regular functions** (*def*)

-has inputs, manipulation and output

-**lamba/anonymous functions** (*lambda*)

-functions that are not formally defined and exist only in place)

-these are to regular functions as list comprehensions are to loops

Condense parts of your scripts to make it more readable:

* A function that represents a single step in your logic

**Another way to condense code is to use list comprehensions**

* A list comprehension to condense a 3 -4 line for-loop into 1 line

Using dicts in place of if else: loops

* {condition: function pointer}

**Generators (aka generator function?)**

-use “yield” rather than “return” which bookmarks the value from the previous iteration rather than starting from the beginning

-*range* is an example of a generator

**Parameters**

foo(param\_a, param\_b=False, \*args, \*\*kwargs)

* Param\_a is most common: a *mandatory parameter*
* Param\_b is optional parameter with a default value
  + Some things to keep in mind: the default value is saved between function calls, and so can be changed if it’s a list/dict
  + Do not use dict or list as optional parameter unless you remember that it is mutable. Can declare as “none” and add a if/then
* *\*args is a variable length list of parameters (between 0... infinity)*
  + *What can we use this for?*
  + *def sum\_args(\*numbers):*
  + *return sum(numbers)*
  + *print sum\_args(23, 42)*
* *\*\*kwargs is keyword arguments-- any new* keyword parameter can be passed in like:
  + foo=[‘a’,’b’,’x’]
  + Then accessible within function like foo = kwargs[‘foo’]
  + What can this be used for?
  + Works well with conditionals (see “hello/goodbye” code example)

Functions are *first class objects* and can be assigned/passed as parameters!

-lambda functions (lambda x: add more condensed logic to things, usually to some sort of iterable operation)

**Pass by Reference vs Pass by Value**

Pass by Reference: we pass a reference (pointer) to the object represented by the parameter

If we declare a list *X*, and pass *X* as a parameter that is *passed by reference*, any changes that we made to the list would persist after the end of the function call.

*Pass by Value*: we pass a copy of the parameter to the function, so that any changes to it do not persist after the function returns

*Which is Python? How can we tell?*

*It is pass by reference, with some exceptions: not all types are mutable (eg: strings of characters are not); assignment does not mutate the list, since the reference is updated to point at a new object.*

Side effects in functions (AVOID THESE)

*What is a side effect?*

*This of this in terms of:*

*Value mutation (e.g. a list changes when running a function)*

*Manipulation of values that weren’t explicitly passed in (e.g. from a global variable)*

*Why should we be wary of them?*

Packages and modules

A *Module* is a discrete python file, ending in ‘.py’ that can be imported via an *import* statement

A *Package* is a collection of modules, identified by a directory with an *\_\_init\_\_.py* file in each directory.