**6.100L Recitation 6 (21 October 2022)**

**Reminders:**

* MQ6 next Monday 10/24
* PS3 halfway hand-in due next Wednesday 10/26

**Lecture 11:**

**Aliasing & Cloning**

* Mutable objects can be changed after they are created.
* What mutable types do we know so far? Lists.
* **Aliasing**: When two variable *names* refer to the same object.
* **Cloning:** Making a copy of an object (typically the safer option).

*Example 1:* Here we are not actually changing the copy of word. Why? Because string are immutable.

word = “the”

word\_copy = word

word += “ bird”

print(word) # “the bird”

print(word\_copy) # “the”

*Example 2:*

a = [1,2,3,4]

b = a

b += [5]

print(b) # [1,2,3,4,5]

print(a) # [1,2,3,4,5]

Now b points to a. Since a list is mutable, if you make changes to b, you will change a.

* For immutable types, “=” creates a new object.
* For mutable types, “=” will assign the new variable to the same object.
* *Why does mutability matter?*
  + Makes your code do unexpected things. For example, you may change a variable you did not want to change.
* *How can I avoid mutability problems?*
  + Make clones, or copies.

List\_copy = list[:]

List\_copy = list.copy()

List\_copy = copy.copy(list)

* Shallow vs Deep copy: shallow copy created new data structurem but actual elements are shared – i.e.top level copy only.

copy.copy(example\_list)  # this is a shallow copy

copy.deepcopy(example\_list)  # this is a deepcopy

* Useful reminder: Don’t change lists while iterating over them!!
* sort vs sorted
  + sort: mutate the list, return nothing
  + sorted: doesn’t mutate the list, return a new sorted list

**Useful List Methods**

* my\_list.copy()  # no mutation - returns copy
* my\_list.reverse()  # mutation
* sorted(my\_list)  # no mutation – returns sorted list
* my\_list.sort()  # mutation
* my\_list.extend([x,y])  # mutation
* my\_list[:]  # makes clone
* my\_list.remove(2) # mutation
* my\_list.pop()  # pops last element - mutation
* my\_list.pop(2)  # pops 3rd element
* my\_list.insert(1, 7)  # inserts 7 in the 2nd position - mutation

**Lecture 12:**

**List Comprehension**

* This is a shorter way to create a new list based on the values of an existing data structure.

# standard method

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

new\_list = []

for x in fruits:  # standard for loop

  if "a" in x:

    newlist.append(x)

# using list comprehension

newlist = [x for x in fruits if "a" in x]

**Default Parameters in Functions**

*Example: here y is a default parameter*

def multiply(x, y=2):

output = x \* y

return output

print(multiply(3)) # outputs 6

print(multiply(3,4)) # outputs 12

**Testing**

Write code that can be broken up into parts and tested easily (inc. comments + assumptions)

3 Class of tests:

* Unit testing - test each function separately
* Regression testing - add tests for bugs as you find them
* Integration testing - high level - does the program do what you want it to

Main 2 testing approaches:

* Black box testing - designed without looking at code, avoids implementer bias, can be reused if implementation changes.
* Glass box testing - use code to guide design of test cases.
* Remember to test edge cases!

**Debugging**

General tips:

* Print out the values of your variables
* Google is your friend if you encounter an error you don’t understand.
* The stack trace shows what line(s) caused the error -- use it!

**Common error messages**

test = [1,2,3]

test[4] # will throw an IndexError since there doesn’t exist an element at index 4

int(test) # will throw a TypeError since lists cannot be converted into integer

# Any error in Python syntax will throw a SyntaxError

MIT OpenCourseWare

https://ocw.mit.edu

6.100L Introduction to CS and Programming Using Python

Fall 2022

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms