**Recitation 4 Notes:**

**Reminders:**

* **No class on Monday**
* **No MQ next week**
* **PS2 half way hand in due next Wednesday**

**Lecture 8: Functions and Scope**

**Functions**

* Functions capture computation within a black box.
* We use them to reuse code and write programs in a more consise way.
* They take in Inputs and return outputs.
* We call the inputs parameters.
* Outputs are outputted using the return statement.

Defining a function:

def count\_letter\_e(my\_word):

count = 0

for letter in my\_word:

if letter == “e”:

count += 1

return count

Calling a function:

print(count\_letter\_e(“hello, this is a test”)

**Print vs return**

* **print:** for the user, just displays a value
* **return**: for the computer and allows you to send values in a function back to other parts of your code. Python’s default return in None and nothing is executed after the return statement.

**Scope**

* Variable assignments are tracked in a **symbol table** or **stack frame** that maps variable names to their values
* When a function is **called**, a new stack frame is created.
* When the function returns, the stack frame pops off/is destroyed
* My python tutor does a good visualization of this <https://pythontutor.com/>.

**Functions as a Parameter**

Example:

def calc(op, x, y):

return op(x,y)

def add(a,b):

return a+b

def div(a,b):

if b != 0:

return a/b

print(“Denominator was 0.”)

print(calc(add, 2, 3))

**Lecture 9 – Lambda Functions, Intro to Tuples & Lists**

**A few additional notes on functions:**

* They have their own type
* Can be passed in as arguments to other functions
* They can be returned as a value from another procedure

**Lambda Functions**

* Anonymous way of writing functions that are not bound to a specific name

E.g.

y = lambda x: x + 5

print(y(4)) # this prints 9 to the console

**Tuples**

* Ordered sequences of objects.
* Syntax: my\_tuple = (1, 2, “test”, 4, “hello”)
* Objects can be of any type.
* They are immutable – i.e. cannot be changed once created.

**Lists**

* Ordered sequence of objects.
* Syntax: my\_list = [1, 2, “test”, 4, “hello”]
* Objects can be of any type.
* They are mutable – i.e. they can be changed once created.

**Common operations on lists and tuples:**

* **Indexing**

my\_list = [1, 2, “test”, 4, “hello]

print(my\_list[0]) # this prints 1

# similarly

my\_tuple = (1, 2, “test”, 4)

print(my\_tuple[2]) # this prints test

* **Slicing**

my\_list = [1, 2, “test”, 4, “hello]

print(my\_list[0:2]) # this prints [1,2]

my\_tuple = (1, 2, “test”, 4)

print(my\_tuple[2:]). # this prints (“test”, 4)

* **Looping over elements –** we can write similar code for both tuples and lists.

my\_list = [1, 2, “test”, 4, “hello]

# this *for loop* loops through each element of my\_list and outputs to console

for elem in my\_list:

print(elem)

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