PIC 16, Winter 2018 – Preparation 1F

Assigned 1/10/2018. To be completed by lecture 1/12/2018.

Intended Learning Outcomes

By the end of this preparatory assignment, students should be able to:

- write Python programs using the equivalents of all their favorite control flow statements from C++ or Java (`for`, `if`, `break`, `continue`),
- use the `range` function to compactly define an arithmetic sequence of numbers,
- write and call functions that accept a variable number of arguments and can have default values for the "formal parameters",
- create lambda expressions to compactly define simple functions, and
- follow recommended coding style, including writing descriptive documentation strings.

Tasks

- ☐ Read 4.1.
- ☐ Using an `if` construct, write a function `compare` that accepts two numerical or string arguments `a` and `b` and returns 1 if `a` is greater than `b`, 0 if `a` equals `b`, -1 if `b` is greater than `a`.
- ☐ Read 4.2 and 4.3. Learn more about slice notation with this video.
- ☐ Write a `for` loop to print out the squares of the integers 1 through 10 (inclusive) like `1 squared is 1`, `2 squared is 4`… `10 squared is 100`.
- ☐ Pass the appropriate arguments to the `range` function such that it returns the list [6, 3, 0, -3, -6]
- ☐ Read 4.4 if you are interested. Personally, I typically don't use `break` or `continue`, so I don't teach them. But it's the same as in C++ or Java.
- ☐ Read 4.5. The phrase "syntactically required" may be misleading; you can always change your code so that `pass` is unnecessary. What they mean is that if, for instance, you try to run code that includes a function or `if` statement before you've added any instructions, like

  ```
  def myFun():
     # There are no statements in this function
  ```

  then you'll get an error "`IndentationError: expected an indented block`". (Try it, and see how adding `pass` in place of the comment fixes it). Anyway, be aware that `pass` exists should you feel that it's needed, but don't look for places to use it.
- ☐ Read 4.6.

  Write a function `swap` that accepts two arguments, `a` and `b`, and returns them in the opposite order. That is, calling

  ```
  a, b = swap(1,2);
  print a, b
  ```

  should produce

  ```
  2 1
  ```

  Yup! Python functions can return multiple variables separated by commas.

  Write a function `sort` that accepts a list of numbers (or strings) and returns them in numerical (or alphabetical) order. Consider two versions of the function: one that changes the list object specified by the argument, and one that does not change the provided list but rather returns a sorted copy. Note that the slicing notation returns a copy of the elements in the list….

Note for Java programmers: Python lists are objects like Java `ArrrayLists`; only the reference to the object is passed into the function. For now, just follow your Java intuition for how everything will behave and it will serve you well.

Note for C++ programmers: lists are like objects created using the `new` keyword in C++. What you're actually passing into the `sort` function is something like a pointer to that object. No copy of the object is made for use inside the function. Effectively, this is similar to passing the list in by reference, because when you change the list inside the function, the changes are made to the list outside the function. Before writing your program, you might want to play with the following example to see how lists behave:

```
def f(x):
    x[1] = 1000

def g(x):
    y = x[:] # creates a copy
    y[1] = 1000
    return y;
```

```
a= [1, 2, 3]
print "Initially, a was", a
f(a)
print "Now, a is",a

b= [1, 2, 3]
print "Initially, b was", b
c = g(b)
print "b is still",b
print "c is",c
```

- Read 4.7.1 – 4.7.3. If you took Java or C++ with me, it seems the text's "actual parameters" are what I would call "arguments" (the values passed into the function) and the text's "formal parameters" are what I would call "parameters" (the variables in the function that take on the values passed into the function).
- Consider the function:

```
def my_fun(a = 10, b = 20):
    print a, b
```

Predict the output of the following:

```
my_fun()
my_fun(1)
my_fun(1,2)
```

How can I call the function to print out `10 30` without passing in the value 10 as an argument? That is, passing in only *one* argument, and leave `a` with its default.

- Write a function `count_args` that accepts any number of input arguments and returns the number of arguments it received, e.g. `count_args(10,2,3,1)` returns `4` and `count_args([10,2,3,1])` returns `1`.
- Ready 4.7.5. You'll get some practice with this in the assignment.
- Read 4.8.