Lab 3b: Python II - Functions, and Flow Control

1 Preliminaries

Download the "Lab03b.py" file into your ENGI128 directory from the web site. Setup your robot, right click on th e file, and select "edit with IDLE".[1] Pull down the "run" menu, and select "Python shell" to open a python interactive prompt. In order to run the file, type:

```
>>> import mcu
>>> mcu.connect()
```

This will connect your computer to your robot. The prompt should change, and now everything you type will go straight to your robot. Continue:

```
python> run Lab03b.py
```

This will run the Lab03b code, and print `all done`.

Getting around IDLE Use these shortcuts to move quickly around IDLE, we have a lot to do!

1. **Alt+P** Previous statement
2. **Alt+3** Block comment.
3. **Alt+4** Block uncomment.
4. **tab (with lines selected)** Indent selection
5. **Shift+tab (with lines selected)** Exdent selection
6. **Ctrl+x** Cut.
7. **Ctrl+c** Copy.
8. **Ctrl+v** Paste.
9. **Alt+tab** Switch applications between IDLE terminal and your file.

2 Variables and Types

Recall that a variable is a name given to a memory location. For example: `i = 5` names some location in memory "`i`", and writes the value 5 to that location. Python on your robots has four primitive variable types:

1. **boolean** can be `True` or `False`
2. **int** Integers. 0, 5, -2938
3. **float** Real numbers (Floating point). 2.1, -1e-12, 3.14159
4. **string** A sequence of characters. `'hello','goodbye'`

And four data structures:

1. **tuple** A combination of variables. Like sticking them together with tape. Cannot be modified after creation.
2. **list** a list of variables: `[1, 2, 3, 4]`. Can be modified after creation. We will cover lists next lecture.
3. **dictionary, set** Data types that we will cover later in the course.

[1]You might be able to double-click on the file to open it with IDLE.

3 Printing

We use the `print` *statement* to print values from a running program, or from inside a function. This is useful for seeing what is going on inside that pretty little robot head:

```
a = 1
b = 2
c = 3
d = 3.14159
this = True
that = False

print a, b, c, d
print this
print that
print alldone
```

4 Functions: `def`

Functions let you write code once, then use it many times. Let's make two functions, `addy` and `multy`:

```
def addy(x, y):
    val = x + y
    return val

def multy(x, y):
    val = x * y
    return val

x = addy(3, 4)
y = multy(3, 4)

print x, y
```

Just like in math, a function takes *arguments* as inputs and *returns* a value as an output. We name the arguments in the definition of the function. Note that we can use the same arguments in two different functions. The *scope* of the arguments is limited to the function they are used in. How does Python know the scope? Look at the indentation. The text that is indented belongs to that function.

Functions do not run when you define them. They need to be *called* from another function, or from the main loop of code. When you call them, you supply the arguments. Python takes the arguments you provide, and makes *new* temporary variables with the names given in the function definition. The function uses these new variables, and returns a value. After the function is complete, the temporary variables are discarded.

Last time, we mentioned the `import` keyword. This imports a *package* of functions. Packages to collect and store functions together. You have already seen two packages, `rone` and `sys`. There will be more over the semester.

5 for

The for loop is used to run a body of code a specified number of times:

```
i = 4
for x in range(0,7):
    print x, i
    i = i + 1
```

We use two special *keywords* here that Python recognizes, `for` and `in`. We also use a built-in function `range()` to create a temporary list of numbers.[2] For loops are useful when you want to repeat something a fixed number of times. But for a more flexible loop, we need to use while.

6 while

Program execution proceeds from top to bottom, unless you change the *flow* of execution. We can do this in many ways. Let's start with a while loop:

```
i = 0
while i < 10:
    print i
    i = i + 1
```

We define the *scope* of while loop by indenting the code to the right. The scope tells Python what code to repeat in the loop. The spaces define the scope; all the indented code belongs to the while loop, and will be repeated. We only use spaces, not tabs for this, and we use four spaces for each level of indent. Python is very picky about the spaces and tabs, if you get it wrong, your code will not run.

```
i = 0
while True:
    print i
    i = i + 1
```

How to stop this thing? Reset the robot with a short tap on the power button. Then use Alt+p to reconnect.

How about loops within loops:

```
i = 0
while i < 5:
   print i
   j = 0
   while j <= i:
       print i,j
       j = j + 1
       break
   i = i + 1
```

[2]Note to advanced Python users, `range` is implemented as `xrange` on the robots.

7 `if/else`

Until now, all of our code ran from top to bottom, maybe with some loops. This is very uninteresting. The `if/else` statement lets the program make decisions about what to run and when. Let's add an `if` statement to our program:

```
i = 0
while i < 10:
    if i > 5:
        print i
    else:
        print -i
    i = i + 1
```

Again, note the spacing used to define the scope of the if statements. Got more to compare?:

```
i = 0
while i < 10:
    if i > 5:
        print i
    elif i < 1:
        print 'Boo!'
    else:
        print -i
    i = i + 1
```

Very exciting. Just in time for Halloween.

Remember the infinite while loop that required us to reset the robot to stop it? We can use an if statement to exit a while loop:

```
i = 0
while True:
   print i
   if i > 15:
       break
   i = i + 1
```

This uses the `break` keyword to exit the while loop.

8 Boolean Variables and Logical Operators

```
this = False
that = False

i = 0
while i < 4:
    print i
    if this:
        print 'this'
    if that:
        print 'that'
    if this and that:
        print 'this and that'
    if this or that:
        print 'this or that'
    i = i + 1

    if (not this) and (not that):
        this = False
        that = True
    elif (not this) and (that):
        this = True
        that = False
    elif (this) and (not that):
        this = True
        that = True
    else:
        this = False
        that = False
```

"You can get with this, or you can get with that"[3] Note the use of parenthesis. These work just like in algebra, it tells the computer what to do first. Use them often.

[3]This is from the '90s group the *Black Sheep*, but you are probably more familiar with the Kia commercial.