1. **Introduction**

**Goal of this Note:**

1. Understand:
2. Functions
3. Conditionals
4. Loops
5. **Functions**

Even though I took CS61A in Fall 2021, I still remember Professor John De Nero threatening in an alarmingly booming voice to burn the computers of those who decided to repeat their code. While the topic he was addressing was data abstraction and recursion (we’ll touch these topics a little next lecture), I believe that data abstraction does not begin with complicated coding, but rather the basics. This brings us to functions.

Functions perform repetitive tasks. They react to some input and produce some output. In other words, if we want to execute a series of statements over and over, we can abstract them away into a function and avoid repeating code.

For instance, imagine that you want to program a function that multiplies a number ‘x’ by 3 and then adds 2 to it. Instead of repeating the act of multiplying 3 and adding 2 each time we desire to carry out this action, we can abstract this action into a function, like the one below:

Diagram

Description automatically generated

Notice that this function takes in an argument x. All functions can take arguments, which is nothing but information you pass to the code inside a function. It is not necessary for a function to have arguments (in which case, we leave empty brackets with no arguments).

Also notice that my function has a return statement. If I do not add this return statement, Python returns None.

When calling my function, I use a call expression. Suppose I am calling the function on a number 10.



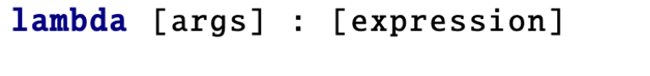
In the call expression above, ‘denji’ is called the operator and 10 is the operand. A call expression applies a function. We always evaluate the operate and then the operand. If there are multiple operands, we evaluate from left to right. =

Functions can also have default arguments where you specify that an argument should have a default value. For example, look at the following function:

Text

Description automatically generated with medium confidence

Notice that all the functions above have a name. Any function starting with a ‘def’ statement has a name. In Python, we can also define small, anonymous functions that do not have name. These are known as lambda functions and are used when you want to pass in another function as an argument. They have the following syntax:



[DEMO on JUPYTER]

1. **Conditionals**

Conditionals are also known as control statements because they literally control the flow of your code. It has the following structure:

If CONDITION:

[Do Something]

elif CONDITION:

[Do Something Else]

else:

[When everything else fails, do this]

[DEMO on JUPYTER]

1. **Loops**
2. While Loops

In Python, while loops are used to execute a block of statements repeatedly until a given condition is satisfied. When the condition becomes false, the loop stops. The statement after the condition is then executed.

A picture containing text

Description automatically generated

In the example above, we instantiate a variable count to be 0. While count is less than 3, we run the while loop. Inside the while loop, we add 1 to the count, print a statement. This process repeats till count becomes 3.

1. For Loops

For loops are used when you want to repeat a process for a number of counts, and you know how many counts this is. Unlike while loop, where you update count variable, for loop does it for you.

Text

Description automatically generated with low confidence

The code above prints ‘i’ for the value of ‘i’ inside the range going from 0 to 4. In more technical terms, for loop iterates through a type of object called iterable.

[DEMO on JUPYTER]

1. **The Go-Do-Your-Homework-Instead Section**
2. *If you want to know about the different types of errors in Python, consider reading the last section of the notes from last year.*
3. *In Lecture 2 notes, I briefly talked about how print statements are of type NoneType. Here’s a challenge for you. Assume that the function ‘denji’ from Section 2 of this note works. What does the following statement return?*

**

*Well, remember that we talked about how call expressions are to be read as operands and then operators. The operator here is the outermost prints statement. We start reading the operands now (from left to right). The first operand ‘print(3, 2, “wow”)’ is itself another call expression, so we evaluate it entirely. So, we print out 3 and then 2 and then “wow’. We now move on to the second operand. Again, ‘print(denji(10)’ is a call expression itself so we now evaluate the statement in its entirety. We call denji on 10. The returned value is 32. We print 32. Since print returns None, both the print statements have returned none. So, the outer operator now reads ‘print(None, None)’ and that’s what we print. Our overall output looks like the following:*

*Text

Description automatically generated*

1. *Now that you have gone through the solution for number 2, try solving the following problem. If you figure it out, PM me the solution and its valid explanation on Slack or come during Office Hours. If it’s right, I’ll pay for your coffee at the Free Speech Café. (Valid only for this week)*

****