Python for Beginners

Lesson 0. A Short Intro
Lesson 1. My First Python Program
Lesson 2. Input from user
Lesson 3. Variables
Lesson 4. If Statements
  How If Statements Work
  Structure
Lesson 5. Simple Loops
Lesson 0. A Short Intro - Python helps Huntington’s Disease

First, if you haven't heard of python already, you might be wondering what it means. In addition to being a snake-like animal, python is also a programming language. Python, is well-known for being an easy language to learn.

Do you know what Huntington's Disease (HD) is? Here is a video that explains the disease. This is a genetic disease, so children of HD patients may be affected too.

Imagine you work in the medical field analyzing the DNA of patients who want to know whether they will come down with Huntington’s disease. Patients get their DNA sequenced and give you the sequence. You analyze the DNA, looking for repeats of the CAG. Depending on how many times CAG is repeated, a simple Python program can tell patients what status they are in. For example, if CAG is repeated 40 times, then the patient will come down with HD.

This program does three tasks:

1. Get the DNA sequence from the patient [HD Task 1]
2. Count how many times CAG is repeated in the sequence [HD Task 3]
3. Print the disease status according to the count [HD Task 2]

We will complete these three Tasks in Python. Let's start with your first python program.
Lesson 1. My First Python Program

Let's start with a simple python program.

**Step 0.** Open Python Shell by going Dashboard -> search for “idle” -> click on IDLE (using Python-2.7). A window should open with a white background and some black text that reads something like:

```python
Python 2.7.3 (default, Apr 10 2013, 06:20:15) [GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> No Subprocess >>>
```
Step 1. Python commands can now be typed into the Python Shell. The interpreter in Python Shell will compile and run those lines of code interactively. Try typing the following command:

```python
print "Alice in Wonderland"
```

Step 2. What happens? Alice in Wonderland should be printed to the screen. The print command just prints anything you tell it to print. Try printing your own messages to the screen. Make sure to include the quotes around your message (explanation to follow). It could be anything like:

```python
print "Oh dear! Oh dear! I shall be too late!"
p
```
Step 2. What did your program say? It should have showed
What is your name?

Step 3. Answer with your name. What happens?

Lesson 3. Variables

Now we can get the input from user, but for more fun, we want "react" to what user has said. For example, if your name is Alice, then we want the program to say Hello, Alice! To do so, your program needs to remember what the user said. A computer program remembers by storing what user has said in a "variable".

You've probably already learned something similar to variables. Have you taken algebra? Remember problems like $4x + 2 = 14$, where you need to solve for $x$? $x$ is like a variable! $x$ has a value, it's storing the data 3 in the equation ($4*3 + 2 = 14$).

In computer programming, we're usually more explicit in naming our variables. We can use any letters, numbers, or an underscore, to name our variables. So, instead of $x$, we might want to have a variable called answer or rabbit or firstName. Then we assign a value to this variable. The value can be of any data type. Let's see this in practice:

Step 0. If you haven't done so already, open a Python Shell.

Step 1. Type the following lines into the command line:

```
firstName = "Alice"
```
print firstName

This might not seem very interesting, but it is! The variable, firstName, is being assigned "Alice". The quotation marks around Alice means that this is a String - a text. The print command is printing the value, what's inside of the variable firstName.

**Step 2.** Let's try adding the last name to the first name. Type the following into the command line:

```python
firstName + " Lewis"
```

What happens? ‘Alice Lewis’ should be printed! The single quotation marks mean that Python understands that Alice Lewis is a text. The python interpreter is using the value of "Alice" for the variable firstName and is adding " Lewis" to "Alice", which turns out to be a new text "Alice Lewis".

**Step 3.** OK, now let's test the "variable" aspect of variables. The Wikipedia definition says that variables are named "variable" because the values can be changed. Try assigning a different value to firstName:

```python
firstName = "Carol"
print firstName
```

firstName is now 'Carol'. Sweet, variables really are variable.
Step 4. This doesn’t seem too useful yet, but let’s begin a little example that will help you realize how powerful this can be. It will be called "Guessing Game". Your program will play the guessing game with you. Your program will think of a number between 1 and 10. You make a guess, and the program will tell you “It’s too high” if your guess is greater than its secret number, and “It’s too low” if your guess is less than its secret number. To begin our example, see if you can create a variable, called secret. Now can you make your program to ask "What is your guess?" Here's what the beginning of the code should look like:

```python
secret = 3 # change to your choice of number between 1 and 10
x = raw_input("What’s your guess?")
```

This assigns whatever guess you typed to the variable x. Enter your guess and type x to verify that your guess is correctly stored.

```python
>>> x
```

It should have single quotation marks around the number you entered.

**HD Task 1.** Write Python code that asks the patient to enter the DNA sequence, and store it in the variable dna. Before we do anything else, let's move this code to a file so we don't have to keep typing it over and over again. Open a new window by going File->New Window in Python Shell, and write the code into the file. Save the file as huntington.py, and save it directly to Desktop.
Step 5. Do you remember what the single quotation marks mean? They mean that Python thinks it is text. For us to compare with the secret number (3 above), we need to change this. Try the following code and see what happens.

```python
>>> guess = int(x)
>>> guess
```

Now there is no single quotation mark! Python understands that variable `guess` has a number. `int` stands for integer, which are whole numbers (no fractions or decimals).

Step 6. Open a new window by going File->New Window in Python Shell, and write the code into the file. Save the file as guessing.py, and save it directly to Desktop. You can run the code in the file by opening it (File->Open) and clicking on Run->Run Module. In your Python Shell, you will see the result as below:

```python
>>> =============== RESTART ===============
>>> 
What’s your guess? 5
```

Lesson 4. If Statements

"OK, great," you might say, "I can create variables. I can change text to numbers. But this isn't very interesting. How does my program say whether the guess is too high or too low?" You could think of it this way: given the answer from the user, if the number is greater than the secret (3 in the example above), then print "It’s too high!". Otherwise, print "It’s too low!". I'll write this in another way:
x = raw_input("What’s your guess? ")
guess = int(x)
if guess > secret:
    print "It’s too high!"
else:
    print "It’s too low!"

Guess what, that’s python code! This is called an "if statement". There are a couple important things to know about if statements. First, you should know how they work. Then you should know the structure. Let's just look at how they work first.

**How If Statements Work**

The code after the word if (guess>secret) is called a *boolean expression*. Huh? Boolean expression is a really strange term. Basically, it means that the expression is either true or false. You can use the following comparison operators to form a boolean expression: ==, !=, <, <=, >, >. Most of these are straightforward, except for maybe the == and != comparison operators. Can you guess what == does? It checks if one piece of data is equal to another. Why not just use 1 equal sign instead of 2, you might ask? Good question. Python already uses one equal sign to mean "assignment". In other words, if you wrote secret = 3, python would think you're assigning the value 3 to the secret variable. A different symbol has to be used for comparison, and that's ==.

Now how about !=. Can you guess what this does? It means **not** equal to. For example, you might want to say "if guess is not equal to secret, do something interesting."
Step 0. Here are some examples of boolean expressions, fill out their meaning (the first one is done for you as an example):

name == "Alice"
Meaning: name is equal to "Alice"

price < 20
Meaning:

grade >= 80
Meaning:

temperature != -50
Meaning:

Going back to the code above, the secret variable is first set to 3. Then, python evaluates the boolean expression guess > secret. In this case it's true, which means runs the code block under the if statement and skips the code under the else statement. If it was false, it would run the code block under the else statement and skip the code under the if statement.
Computer Science for San Francisco Youth

**Structure**

The first thing note about the structure: you need to place a colon (:) after the if statement. Look at the code again.

```python
if guess > secret: #<- colon here
    print "It’s too high!"
else: #<- colon here
    print "It’s too low!"
```

See the colons? If you don't include these, there will be an error when you try to run your code. The colons tell python that the if and else statements are over and the next lines of code are the code blocks.

In the if statement above, there are 2 *blocks* of code. One block is under the first *if* statement and another block is under the *else* statement. Notice that the code blocks are indented. I used a tab here, but you could also use spaces. The important thing is that all lines of code in the same code block are indented the exact same amount.

In the example above, each block has only 1 line of code, but that's not always the case. There could be several lines of code in each block. Here's an example:

```python
if guess > secret:
    print "It’s too high!"
    print "Try a smaller number."
else:
    print "It’s too low!"
```
Like I mentioned earlier, all code in the block needs to be indented the same amount.

**Step 0.** Now that you know the structure, try entering the following code into guessing.py.

```python
import random
secret = random.randint(1,10) #This line picks a number between 1 and 10
x = raw_input("What’s your guess? ")
guess = int(x)
if guess > secret:
    print "It’s too high!"
else:
    print "It’s too low!"
```

**Step 1.** There’s actually one more type of statement that you can use: the *elif* statement. Any idea what this might stand for? *else if.* This gives us even more options for our *if* statements. Here’s an example:

```python
import random
secret = random.randint(1,10) #This line picks a number between 1 and 10
x = raw_input("What’s your guess? ")
guess = int(x)
if guess > secret:
    print "It’s too high!"
elif guess < secret:
    print "It’s too low!"
else:
```
print "You won!"

The elif statement is placed between the if and else statements and includes a boolean expression like the if statement. During the execution of the program, the boolean expression in the if statement is first tested. If it is true, the code block under the if statement is executed and the rest is skipped. If it is false, the boolean expression in the next elif statement is tested. If that boolean expression is true, the code in the code block will be executed and the rest is skipped. If none of the boolean expressions are true, only the code in the else statement is executed.

**HD Task 2.** Huntington’s Disease status depends on the CAG repeat count as below. Add if-elif-else statement to huntington.py from HD Task 1 to print right messages for patients according to the number stored in variable `count`. The first if statement is provided.

```python
if (count<26):
    print "Normal. This patient is unaffected."
```

<table>
<thead>
<tr>
<th>Repeat count</th>
<th>Classification</th>
<th>Disease status</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;26</td>
<td>Normal</td>
<td>Unaffected</td>
</tr>
<tr>
<td>26–35</td>
<td>Intermediate</td>
<td>Unaffected</td>
</tr>
<tr>
<td>36–39</td>
<td>Reduced Penetrance</td>
<td>May or may not be Affected</td>
</tr>
<tr>
<td>&gt;=40</td>
<td>Full Penetrance</td>
<td>Affected</td>
</tr>
</tbody>
</table>
Lesson 5. Simple Loops

Imagine that you want to play the guessing game that you made. It will be boring that you have to click on Run->Run Module every time you want to make a guess. Computers are perfect for doing the repetition for you. Let's start with printing the same thing multiple times.

**Step 1.** Try the following range statement, what happens?

```python
range (0,5)
```

**Step 2.** Try the following for-loop statement, what happens?

```python
>>> for i in range (0,5):
...       print i #<= don’t forget to tab before print
...
0
1
2
3
4
```

**Step 3.** Fill in the numbers below to make the guessing game ask (repeat) 3 times.

```python
import random
secret = random.randint(1,10) #This line picks a number between 1 and 10
for i in range # Fill in the numbers
x = raw_input("What’s your guess? ")
guess = int(x)
if guess > secret:
    print "It’s too high!"
elif guess < secret:
    print "It’s too low!"
else:
    print “You won!”

Later, we will use the for-loop statement to repeat the search for CAG in the DNA sequence. The DNA sequence can be very long, and the for-loop statement will help finding all CAGs and counting them.

**Lesson 6. Finding CAG**

How do you create a nickname for a friend? Many people use a short form of their names as nicknames. For example, Elizabeth could be Eli, Liz, Liza, or Beth. Python is very good at finding all possible nicknames. Let’s try this:

**Step 0.** Open a Python Shell as usual.

**Step 1.** At the command line, type the following:

name = “Elizabeth”
name[0:3]

What is printed? It should say Eli. All programming languages start counting at 0, unlike
Humans who start at 1. name[0:3] gives you a short name starting from the first letter (0) and ending before the 4th letter (3). Let’s try some more.

name[1:4]
name[1:5]
name[5:9]

Imagine the DNA sequence is “ATTCAG” stored in variable dna. What will dna[3:6] be print?

**Step 2.** To find all CAGs stored in the DNA sequence, we need to look at dna[0:3], dna[1:4], dna[2:5], dna[3:6], etc. How do we know when we can stop? Try the following code.

dna = “ATTCAG”
len(dna)

6 should have been printed. Let’s try some more. What values are printed?

dna = “ATTCAGCCCC”
len(dna)
dna = “ATTCAGCAGCAG”
len(dna)

So, we need to look at dna[0:3], dna[1:4], dna[2:5], dna[3:6], ...
dna[len(dna):len(dna)+3].

Step 3. Remember the for-loop statement with range(0,5)? What will be printed from the following for-loop?

```python
dna = "ATTCAG"
for i in range(0, len(dna)):
    print dna[i:i+3]
```

Step 4. Remember the if statement with == symbol? What will be printed from the following for-loop?

```python
dna = "ATTCAG"
for i in range(0, len(dna)):
    if dna[i:i+3] == "CAG"
        print "CAG"
```

Step 5. Ok, almost there. Remember the if statement with == symbol? What will be printed from the following for-loop? Add more CAGs to dna and see what happens.

```python
dna = "ATTCAG"
count = 0
for i in range(0, len(dna)):
    if dna[i:i+3] == "CAG":
        count = count+1
print count
```

HD Task 3. Use the Python code from HD Task 1 and HD Task 2 to complete huntington.py, Huntington’s Disease program. Good job!!