Lab 7: Teaching Computers to Make Decisions

Up to now, all the programs you have written in this class involved telling the computer exactly what to do. However, much of the true power of computing lies in being able to get the computer to decide what to do in different circumstances. Of course, you still have to tell it what to do in each circumstance, but once those instructions are written, the computer makes decisions without explicit input from you.

This lab will introduce several pieces of syntax, which might seem overwhelming. Fortunately, if you forget something, you can use the pull-down menus in CoCalc to get an example of what you want. You can then modify the example to suit your needs.

You can also find documentation on a command in SageMath by typing the command, a question mark, and pressing "Run" or Shift+Enter (ex: “if?”).
If statements

The most important way to get computers to make decisions is called an if statement. It tells the computer what to do if some condition is met.

```python
>>if temp > 90:
    print "It's hot!"

>>if n < 5:
    n = n+1
```

**Exercise 1.** Write and test an if statement that prints the word “jackpot” if money>1000000. (In order to run this example and the ones above, you first need to assign a value to the appropriate variable, in this case money.)

You can also tell the computer what to do if the condition is not met.

```python
>>n=4
>>if n<5:
    n = n+1
    print n
else:
    n = n+2
    print n
```

Any actions nested under else are only carried out if the first condition is not met.

**Exercise 2.** Run the above example for n=4 and n=6. Remember to assign a value to n before evaluating the cell.

**Exercise 3.** Write and test an if statement that prints “First president” if person=="George Washington" and “Not first president” otherwise. (The double equal sign is used to determine if one value is equal to another. Be careful not confuse it with the single = sign, which assigns a value to a variable.)

It’s also possible to specify more than one alternative to the situation in the original if statement. In Python and Sage, this is done using the keyword elif, which is short for “else if”.

```python
>>if n<5:
    n = n+1
    print n
elif n>7:
    n = n-1
    print n
else:
    n = n+2
    print n
```
You can have as many `elif`'s as you want after an initial `if`. However, the whole collection of conditions and actions, termed an *if-else statement* or *if-then-else statement*, must end with `else`.

**Exercise 4.** Write and test an if-else statement that prints, “It’s hot” if `temp>90`, “It’s cold” if `temp<70`, and “Not bad” otherwise.

**Exercise 5.** Modify the above statement to reflect the opinions of someone from Alaska rather than Los Angeles.

If statements can be usefully combined with other programming structures, such as for loops. You can place an if statement inside a for loop or a for loop inside an if statement.

**Example 1.** This code prints “long word” if a word has more than 8 letters and “not very long word” otherwise.

```python
words = ['elephant', 'mongoose', 'emu', 'crocodile', 'chimp']
for word in words:
    if len(word)>8:
        print "long word"
    else:
        print "not very long word"
```

**Exercise 6.** Extend this example to print “short word” if a word has 4 or fewer letters.

**While loops**

A for loop repeats an action for every item in a list. However, sometimes an action needs to be repeated until some condition is met. For example, the loop below prints numbers until it finds one whose square root is 2.5 or greater.

```python
n = 0
while sqrt(n) < 2.5:
    print n
    n = n+1
```

Notice that we needed to give the variable `n` a starting value. Also, note that the last line of the loop increases `n` by 1. If that line wasn’t there, we would have an *infinite loop* – a loop that never ends. This is a common mistake in programming. To interrupt an infinite loop in Sage, click the Stop button (the orange square), fix your code, and then run it again, re-evaluating any cells that contain necessary code.

Sometimes, either a for loop or a while loop can perform a task. For example, this while loop prints the numbers from 1 to 10, which could be done with a for loop. (How?)

```python
n = 1
while n <= 10:
    print n
    n = n+1
```
>> print n
>> n = n+1

**Exercise 7.** Type in and run this example. Write a comment on every line that explains what it does.

**Exercise 8.** Write a program that uses a while loop to print a phrase 10 times. HINT: Keep track of how many times it has already been printed.

**Other useful things to know**

In either an if-statement or a while loop, you can use **and** to specify multiple conditions that must all be met.

```python
>>> if looks=="duck" and quacks=="duck":
    print "duck"
```

Also, you can use `!=` for “not equal”, `>=` for `≥`, `<=` for `≤`, and `not` (some condition) for the opposite of a condition, as in `not color == "red"`.

**Exercise 9.** Make up examples of if statements or while loops using two of the things described in this section.

**Reusing code**

Think about the Sage command `plot`. It takes inputs (a function, a plotting range, and various plotting options) and gives an output. The proper name for such commands is **functions**. In this course, we’ll call them Python functions to distinguish them from mathematical functions. Python functions can include loops, if statements and pretty much any other kind of code. Code that is not a function is sometimes called a **script**.

Python functions always have the form of the following non-functioning “skeleton”:

```python
def functionName(arguments):
#Do stuff
    return output
```

A very simple function could just use

```python
def functionName(arguments):
    return output
```

In this function skeleton, **def** is a Python keyword that marks the beginning of the function. (If it looks familiar, you saw it while working with interactives. Interactives are just a nice way of providing input to functions.) The keyword **return** tells the function to give us the specified output and stop. Some functions, particularly those used for graphics and animation, don’t use **return**, but most others do.
Here is a simple Python function that takes a number and squares it.

```python
def sqr(num):
    num2 = num**2 #Squares the number
    return num2 #Gives the output
```

You could also write this in a shorter format.

```python
def sqr(num):
    return num**2 #Squares the number and outputs it
```

Here is another example: a function that multiplies its argument by 5 and raises it to the third power. Note that this calculation is done in one step.

```python
def fivecube(n):
    result = (5*n)**3
    return result
```

The body of a function must be indented relative to the top line. If you type the function directly in Sage, the indentation will usually be inserted automatically. If necessary, use tabs to indent.

**Exercise 10.** Write a Python function that cubes its argument and multiplies the result by 19.

**Exercise 11.** Write a Python function of your own choosing that includes an if statement. Include comments explaining what each line does. Then, show the function’s output for several inputs. (HINT: To display the outputs of several commands on one line, separate the commands with semicolons.)

The great virtue of functions is that they allow you to reuse code. Instead of copying and modifying code every time you want to use it, you can write a function. Then, you can use the function with different inputs, just like you would with `plot`. Using a function is referred to as calling the function.

**A Taste of Bioinformatics**

While this class focuses on dynamical modeling and simulation, programming is also a critical skill in genetics and molecular biology, especially now that large datasets are easy to obtain. In this part of the lab, you will take a look at some uses of programming in genetics.

A DNA sequence can be represented as a sequence of As, Cs, Gs and Ts. The simplest way to work with such sequences in Sage is to make them into character strings. Character strings, usually just called *strings*, are made by listing characters in quotation marks. For example, "cat" is a string. Most of the things you know about working with lists also apply to strings, including the `len` function and the use of square brackets to look at individual entries. However, instead of using `append` to add a single item onto a string, you have to use `+`. (`+` also works for lists.)

```bash
>>>"cat"+"s"
"cats"
```
The complement of a DNA sequence is another sequence that gives the nucleotides that the original sequence would pair with. For example, if the original sequence is GACT, its complement is CTGA.

**Exercise 12.** Write a script that will find the complement of the sequence AGGCTA. (HINT: It is possible to put an if statement inside a loop.)

**Exercise 13.** Write a function that will find the complement of any sequence.

We can compare sequences of different genes or of the same gene in different species by computing what fraction of their nucleotides they have in common. Here, you will work with real sequences for insulin from four different species: humans, chimpanzees, pigs and mice. (Insulin from pigs was used by diabetes patients before human insulin from genetically modified bacteria was available.) These sequences have been aligned to correct for the effects of nucleotide additions and deletions, which can throw off the whole comparison. Thus, they contain dashes where such a mutation is thought to have taken place.

**Exercise 14.** Find what fraction of the human and chimpanzee nucleotides are identical, including dashes in the comparison.

**Exercise 15.** Write a function that can compare any two sequences. Use it to compare all the sequences to each other. Write a summary of your results or make a table using the table function. (An example of the use of table is table([[“a”,”b”,”c”], [“cat”,”dog”,”rabbit”]]) .)