6.189 Day 2

Name & Athena username:

Tear off the first three pages of this handout and turn them in at the start of lecture tomorrow. Print out and staple your code files (with your name in comments at the top) for the last three exercises to it.

Readings

How To Think Like A Computer Scientist, chapter 4, sections 1, 2, 4-7; chapter 6.2.

Exercise 2.0 – New Operators

Open up IDLE and play around with the new operators showed today in class. Make sure that you understand how to use them and what they are used for! The operators `==, !=, <, >, <=, >=` are called *relation operators*. They work on all types, not just numbers, and return a *Boolean* (True/False) value. Remember, if you are using Booleans, to capitalize True and False! Here's an example shell session; try other examples you can think of.

```
>>> 5 >= 7
False
>>> 'abc' != 'def'
True
>>> x = 'abc'
>>> x == 'abc'
True
>>> a = True
>>> b = 5 < 7
>>> a == b
True
```

That last example is strange! Try to understand what's going on there, and ask if you're confused. Next, the operators `+=, -=, *=, /=` change the value of a stored variable in a quicker way. In the following example, we add 6 to a variable in two different ways; note that we get the same result! Try using all of these operators in your interpreter window before moving on.

```
>>> x = 5
>>> x = x + 6
>>> print x
11
>>> y = 5
>>> y += 6
>>> print y
11
```

Exercise 2.1 – Boolean operators

Boolean operators can seem tricky at first, and it takes practice to evaluate them correctly. Write the value (`True` or `False`) produced by each expession below, using the assigned values of the variables a, b, and c. Try to do this without using your interpreter, but you should check yourself when you think you've got it.

```
a = False
b = True
c = False
```

1. `b and c`

2. `b or c`

3. `not a and b`

4. `(a and b) or not c`

5. `not b and not (a or c)`

Hint: Work from the inside out, starting with the inner-most expressions, like in arithmetic.

Exercise 2.2 – Blackjack

The purpose of this exercise is to understand conditionals. Consider the following fragment of code, showing a *basic* strategy for Blackjack (yes, there are more complicated ones, but bear with me for this exercise). The goal of the game is the total value of your cards to be higher than that of the dealer's without going over 21 (also known as "busting"). If the total initial value of your cards is 21, it is called blackjack and you automatically win unless the dealer also has a blackjack in which case the hand is a tie.

```
card1 = ____
card2 = ____
dealer_card = ____

total = card1 + card2

if total == 21:
    print 'Blackjack'
elif total >= 17:
    print 'Stand'
elif total >= 12:
    if dealer_card > 6:
       print 'Hit'
    else:
       print 'Stand'
elif total > 9 or (total == 9 and dealer_card <= 6):
    print 'Double then hit'
else:
    print 'Hit'
```

For each of the following card values, write down the output that would be generated. Do this without running the code. It is an important skill to be able to understand what a piece of code does without running it.

1.
   ```
   card1 = 3
   card2 = 8
   dealer_card = 4
   ```

2.
   ```
   card1 = 9
   card2 = 8
   dealer_card = 10
   ```

3.
   ```
   card1 = 11
   card2 = 10
   dealer_card = 10
   ```

4.
   ```
   card1 = 3
   card2 = 5
   dealer_card = 4
   ```

5.
   ```
   card1 = 5
   card2 = 9
   dealer_card = 8
   ```

Exercise 2.3 – Understanding loops

For each of the following fragments of code, write what the output would be. Again, do this without running the code (although feel free to check yourself when you're done).

1.
```
num = 10
while num > 3:
   print num
   num = num - 1
```

2.
```
divisor = 2
for i in range(0, 10, 2):
   print i/divisor
```

3.
```
num = 10
while True:
   if num < 7:
      break
   print num
   num -= 1
```

4.
```
count = 0
for letter in 'Snow!':
   print 'Letter #', count, 'is', letter
   count += 1
```

Exercise 2.4 – Buggy loop (aka Find The Bug!)

Consider the following program (again, try to do this exercise without running the code in IDLE!):

```
n = 10
i = 10


while i > 0:
   print i
   if i % 2 == 0:
      i = i / 2
   else:
      i = i + 1
```

1. Draw a table that shows the value of the variables n and i during the execution of the program. Your table should contain two columns (one for each variable) and one row for each iteration. For each row in the table, write down the values of the variables as they would be at the line containing the print statement.

2. What is problematic about this program? Suggest one way to improve its behavior.

Exercise 2.5 – Rock, Paper, Scissors

In this exercise, you are going to practice using conditionals (if, elif, else). You will write a small program that will determine the result of a rock, paper, scissors game, given Player 1 and Player 2's choices. Your program will print out the result. Here are the rules of the game:

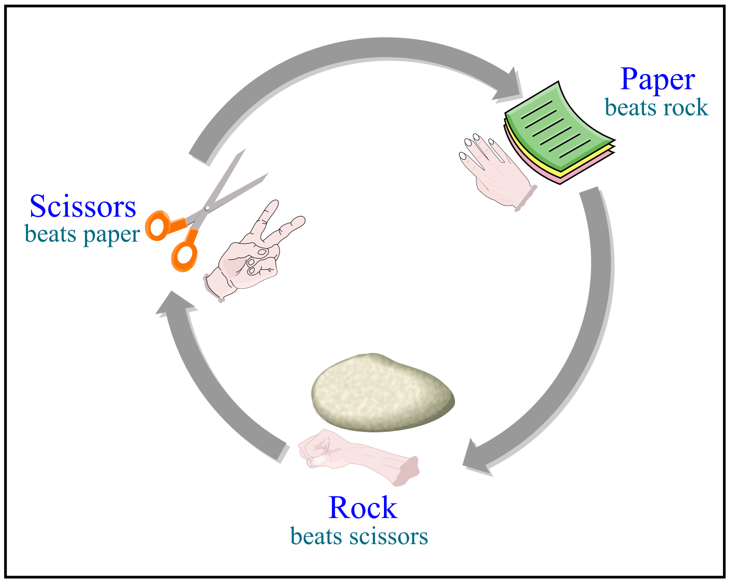

Figure by MIT OpenCourseWare.

1. First create a truth table for all the possible choices for player 1 and 2, and the outcome of the game:

Player 1	Player 2	Result
Rock	Rock	Tie
Rock	Scissors	Player 1

2. Create a file rps.py that will generate the outcome of the rock, scissors, paper game. The program should work as follows:

   ```
   Player 1? rock
   Player 2? scissors
   Player 1 wins.
   ```

   The only valid inputs are rock, paper, and scissors. If the user enters anything else, your program should output "This is not a valid object selection". Use the truth table you created to help with creating the conditions for your if statement(s).

   Note If you have a long condition in your if statement, and you want to split it into multiple lines, you can either enclose the entire expression in parenthesis, e.g.

   ```
   if (player1 == 'rock' and
      player2 == 'scissors'):
      print 'Player 1 wins.'
   ```

   Or, you can use the forward slash symbol to indicate to Python that the next line is still part of the previous line of code, e.g.

   ```
   if player1 == 'rock' and\
      player2 == 'scissors':
      print 'Player 1 wins.'
   ```

   Use whichever form you feel comfortable using. When you are done coding *and testing!*, print a copy of the file and turn it in. Make sure your name is in the comment section of your program, in the same manner as explained in the Day 1 exercises.

Exercise 2.6 – While Loops

Create a file called loops.py for both parts of this exercise. If you didn't do yesterday's optional problem, you should look at the **Note** about `raw_input` versus `input`. You will want to use `input` for this exercise.

1. Write a program that asks the user for a number, and prints a countdown from that number to zero. What should your program do if the user inputs a negative number? As a programmer, you should always consider "edge conditions" like these when you program! (Another way to put it- always assume the users of your program will be trying to find a way to break it! If you don't include a condition that catches negative numbers, what will your program do?)

2. Write a program that will ask the user to enter a number that is divisible by 2. Give the user a witty message if they enter something that is not divisible by 2- *and make them enter a new number*. Don't let them stop until they enter an even number! Print a congratulatory message when they *finally* get it right.

When you are done, print a copy of the file and turn it in. Make sure your name is in the comment section of your program.

Exercise 2.7 – Secret Messages

This exercise is tricky! Be sure to ask the LAs for help if you need them!

The goal of this exercise is to write a cyclic cipher to encrypt messages. This type of cipher was used by Julius Caesar to communicate with his generals. It is very simple to generate but it can actually be easily broken and does not provide the security one would hope for.

The key idea behind the Caesar cipher is to replace each letter by a letter some fixed number of positions down the alphabet. For example, if we want to create a cipher shifting by 3, you will get the following mapping:

```
Plain:   ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher:  DEFGHIJKLMNOPQRSTUVWXYZABC
```

To be able to generate the cipher above, we need to understand a little bit about how text is represented inside the computer. Each character has a numerical value and one of the standard encodings is ASCII (American Standard Code for Information Interchange). It is a mapping between the numerical value and the character graphic. For example, the ASCII value of 'A' is 65 and the ASCII value of 'a' is 97. To convert between the ASCII code and the character value in Python, you can use the following code:

```
letter = 'a'

# converts a letter to ascii code
ascii_code = ord(letter)

# converts ascii code to a letter
letter_res = chr(ascii_code)

print ascii_code, letter_res
```

Start small. Do not try to implement the entire program at once. Break the program into parts as follows:

1. Create a file called cipher.py. Start your program by asking the user for a phrase to encode and the shift value. Then begin the structure of your program by entering in this loop (we'll build on it more in a bit):

   ```
   encoded_phrase = ''

   for c in phrase:
      encoded_phrase = encoded_phrase + c
   ```

   What does this loop do? Make sure you understand what the code does *before* moving on!

2. Now modify the program above to replace all the alphabetic characters with 'x'. For example:

   ```
   Enter sentence to encrypt: Mayday! Mayday!
   Enter shift value: 4
   The encoded phrase is:  Xxxxxx! Xxxxxx!
   ```

   We are going to apply the cipher only to the alphabetic characters and we will ignore the others.

3. Now modify your code, so that it produces the encoded string using the cyclic cipher with the shift value entered by the user. Let's see how one might do a cyclic shift. Let's say we have the sequence:

   ```
   012345
   ```

   If we use a shift value of 4 and just shift all the numbers, the result will be:

   ```
   456789
   ```

   We want the values of the numbers to remain between 0 and 5. To do this we will use the modulus operator. The expression x%y will return a number in the range 0 to y-1 inclusive, e.g. 4%6 = 4, 6%6 = 0, 7%6 =1. Thus the result of the operation will be:

   ```
   450123
   ```

   Hint: Note that the ASCII value of 'A' is 65 and 'a' is 97, not 0. So you will have to think how to use the modulus operator to achieve the desired result. Apply the cipher separately to the upper and lower case letters.

   Here is what you program should output:

   ```
   Enter sentence to encrypt: Mayday! Mayday!
   Enter shift value: 4
   The encoded phrase is:  Qechec! Qechec!
   ```

   When you are done, print a copy of the file and turn it in. Make sure your name is in the comment section of your program.