Homework Review!

Let's work through

- the "odds and evens" problem
- the string formatting exercise

Conditions and Loops

Conditions

We often want to use logical statements (a test of whether or not some condition holds) to determine what our program should do.

Here is some pseudocode (a framework of what we want our code to do, but not written in real code yet):

```
if condition1 is true
    then do x
if condition2 is true instead
    then do y
otherwise
    do z
```

Boolean Values Refresh

`bool` objects allow us to decide which conditions are or are not met

- Each condition is evaluated as `True` or `False`
- These are the basis for determining which branch in our logic should be followed

If, Else Statements

In programming, conditions are expressed through the keywords `if` and `else` (and also `elif` in Python)

Let's write code based on the relationship between `a` and `b`.

```
if a<b:
    print("Smaller!")
elif a>b:
    print("Larger!")
else:
    print("Equal!")
```

Set `a=12` and `b=100`. What happens? What if `b=12`?

Making a Good Condition

- `if`, `else` statements are order dependent!
- The logic must be clear
  - Shouldn't compare strings to integers, for example
- You must enumerate all possible outcomes. If not, your code might surprise you!
  - Remember, computers are stupid (like, REALLY stupid), and only do what they are told
  - This is the "Garbage in, garbage out" principle

Introducing Loops

Sometimes, we need to execute code multiple times, or need to run that code until some condition is met.

- `for` loops allow us to iterate through code a fixed number of times
- `while` loops allow us to repeat code until a predetermined condition is met
- functions allow us to recycle code of form!

For Loops

The word `for` is reserved in Python to denote the start of a **for loop**.

```
for i in [1,2,3,4,5,6,7,8,9,10]: # repeat code below for
    print(i)                     # each value in the list
```

This code will run one time for each value in the list, and each iteration will assign the next value in the list to the variable `i`.

- We then are told to print `i`, so that each number between 1 and 10 is printed in order.

Exercise

Write a `for` loop that will square each number between 1 and 10, and print the value.

Exercise

Write a `for` loop that will square each number between 1 and 10, and print the value.

```
for i in range(1,11): # range is an iterable, with a lower
    print(i**2)       # bound included, and upper bound
                      # excluded
```

Exercise

Write **nested** `for` loops (a `for` loop inside of a `for` loop) that will square each number in a matrix stored as a list of lists. Use the following matrix:

```
myMatrix=[[1,2,3],
          [4,5,6],
          [7,8,9]]
```

Exercise

Write **nested** `for` loops (a `for` loop inside of a `for` loop) that will square each number in a matrix stored as a list of lists. Use the following matrix:

```
myMatrix=[[1,2,3],
          [4,5,6],
          [7,8,9]]
```

```
for i in range(3): # 3 is the number of elements in a row
    for j in range(3): # 3 elements per column
        myMatrix[i][j]**=2 # row THEN column
```

Note: `*=` means to multiply and then update the value, and `**=` means exponentiate then update value

Some Shorthand

We can also write for loops as **list comprehensions** in order to quickly create lists:

```
myList = [x**2 for x in range(1,11)]
```

This will provide us with the same list as

```
myList = []
for i in range(1,11):
    myList.append(i**2)
```

More Shorthand

- `+=` means to add the second value, and assign the sum to the original variable
- `-=` means to subtract the second value, and assign the difference to the original variable
- `*=` means to multiply the values, and assign the product to the original variable
- `/=` means to divide the first value by the second, and assign the ratio to the original variable
- `**=` means to exponentiate the value (with the second number being the exponent), and assign the product to the original variable

While Loops

`while` loops require a logical statement

- **While** the statement is true, the loop continues to execute
  - When the condition no longer holds, the loop terminates
- This is typically where programmers create "infinite loops"
  - Always remember to update your condition variable in each iteration!

While Loops

```
x=1
y=11
while (x<y):
    print(x)
    x+=1
```

This will print `x` every time the loop executes until `x` is no longer less than `y` (will NOT print when the two are equal).

Is the line `x+=1` necessary? Discuss with your neighbor.

Lab Time!

Complete the problems in PyCharm! 😀