Intro to Python Class 2

Review

- Arithmetic and variables
- Data types
- Text editor, command line, and python shell

What we will cover the rest of today

- Boolean Expressions
- Conditionals
- Lists
- Loops

Boolean Expressions

We can tell the computer to compare values and return True or False. These are called Boolean expressions

- Test for equality by using `==`. We can't use `=` because that is used for assignment
- Test for greater than and less than using `>` and `<`

```
a = 5
b = 5
print(a == b)
c = 3
print(a == c)
print(c < a)
```

Boolean Expressions continued

The following chart shows the various Boolean operators

`a == b`	a is equal to b
`a != b`	a does not equal b
`a < b`	a is less than b
`a > b`	a is greater than b
`a <= b`	a is less than or equal to b
`a >= b`	a is greater than or equal to b

```
a = 3
b = 4
print(a != b)
print(a <= 3)
print(a >= 4)
```

Remember: Equals does not equal "equals equals"

Lists

A list is an ordered collection of elements

In Python, a list is defined using `[ ]` with elements separated by commas, as in the following example

```
words = ['list', 'of', 'strings']
```

A list can, but doesn't have to be of all one type.

A list of one type is homogenous as opposed to a list of multiple types, which is heterogeneous.

```
words = [0, 'list', 'of', 3, 'strings', 'and', 'numbers']
```

Appending to list

Lists have several methods, the most useful of which is `append`

A list can be created as an empty list and have values added to it with `append`

```
to_do_list = []
to_do_list.append('buy soy milk')
to_do_list.append('learn python')
print(to_do_list)
```

Therefore, lists are mutable

This means that a list can change values
during the duration of a program

Indexing

An element can also be obtained from a list through indexing

This allows us to obtain an element if we just want one specific value.

To index on a list, follow it immediately with `[index]`.

```
numbers = [10, 20, 30]
print(numbers[0])
```

Lists (and other data types) start at the number 0 and count up.

Length and More Indexing

You can get the length of a list with len(list)

You can also use the length of the list to index the last element.

```
to_do_list = [
    'learn python', 'read email', 'make lunch',
]
print(len(to_do_list))

print(to_do_list[0])
print(to_do_list[2])

print(to_do_list[len(to_do_list)])
print(to_do_list[len(to_do_list) - 1])
```

An IndexError results if an index exceeds the length of the list minus 1

Let's Develop It

Take the list below and try to do the following:

Append to the list a chore you have to do

Use indexing to find the first and third chores on the list

Find the length of the list

Use indexing to find the last chore in the list

```
['Make the bed', 'Cook dinner', 'Feed the dog', 'Buy groceries']
```

Conditionals

When we want different code to execute dependending on certain criteria, we use a conditional

We achieve this using if statements

```
if x == 5:
    print('x is equal to 5')
```

Conditionals

We often want a different block to execute if the statement is false.
This can be accomplished using else

```
if x == 5:
    print('x is equal to 5')
else:
    print('x is not equal to 5')
```

Indentation

In Python, blocks begin when text is indented and ends when it returns to the previous indentation

Indentation is very sensitive in python. Let's write a python file that practices indentation and conditionals

Write this into your text editor and save the file as class2.py in your gdi-intro-python folder.

```
print("It's your birthday!")
answer = input("How old are you? ")
age = int(answer)  # convert to an integer from a string
if age < 21:
    print("You may not have a beer, but here's some juice!")
else:
    print("Here's some beer!")
print("Happy birthday!")
```

Chained conditionals

Conditionals can also be chained

Chained conditionals use `elif` as an additonal check after the preceeding `if` predicate was False. For example:

```
if x < 0:
    print("x is negative")
elif x > 0:
    print("x is positive")
else:
    print("x is 0")
```

Nested conditionals

Conditionals can also be nested

Nested conditionals occur inside of other conditionals, and are indented over once more. When the code block is complete you move back.

Edit your python file to contain the following:

```
print("It's your birthday!")
answer = input("How old are you? ")
age = int(answer)
if answer < 21:
    print("You may not have a beer, but here's some juice!")
else:
    beers = input("How many beers do you want?")
    beers = int(beers)
    if beers > 3:
        print("Oops, you're drunk!")
    elif beers > 1:
        print("You got a little tipsy")
    else:
        print("Looks like you're the designated driver")
print("Happy birthday!")
```

Let's Develop It

Write a program that uses if statements to determine what to do given some user input

The code below is an example:

```
health = 100
print("A vicious warg is chasing you.")
print("Options:")
print("1 - Hide in the cave.")
print("2 - Climb a tree.")
input_value = input("Enter choice:")
if input_value == "1":
    print("You hide in a cave.")
    print("The warg finds you and injures your leg with its claws")
    health = health - 10
elif input_value == "2":
    print("You climb a tree."
    print("The warg eventually looses interest and wanders off")
else:
    print('Invalid option.')
```

Iteration

It is often useful to perform a task and to repeat the process until a certain point is reached.

The repeated execution of a set of statements is called iteration

One way to acheive this, is with the while loop.

```
user_input = "n"

while user_input != "y":
    user_input = input("would you like to quit? (y/n) ")

print("quitters never win!")
```

As long as the while statement evaluates to True, the code block beneath it is repeated.

While loops

Consider the following example that uses iteration with a while loop

```
input_value = input("Enter a positive integer:")
user_number = int(input_value)
while user_number > 0:
    print("The user's number is still positive, let's subtract 1")
    user_number = user_number - 1
print("User's number is no longer positive.")
```

This implementation does not work for negative numbers. Why?

For loops

It is also useful to loop through a collection of elements, visiting each one to do some work, then stopping once all elements are processed.

This can be accomplished with a `for` loop

First, we need a collection. We create a `list` of numbers to loop over. This is called `numbers` in the following example

```
shipping_cost = 2.5
prices = [3, 4, 5.25]
costs = []

for price in prices:
    costs.append(price + shipping_cost)

print(costs)
```

Range

What do you think the following functions output?

```
range(5)
```

```
range(5, 10)
```

```
range(10, 20, 2)
```

```
range(20, 8, -2)
```

For loops continued

Let's examine the example carefully

```
for number in range(5):
    print("The current number is:")
    print(number)
```

This for loop has three parts:

- The collection to loop over - range(5)
- The name WE give each element when the loop begins again - number
- The block of statements to execute with the element - The two print statements

Variable name in for loops

In our last example, we named our variable "number".

We could have chosen any word, and the computer would process the code.

This is a great example of the computer *processing*, not *understanding*.

```
for moose in range(5):
    print("The current number is:")
    print(moose)
```

Let's Develop It

Example, Let's iterate through the following list:

```
["Mary", "had", "a", "little", "lamb"]
```

For each word, check if the length is 4 and if so print it out,

Hint: Just like lists, the len functions works on strings. len('hello') returns 5.

For example:

```
Mary
lamb
```

What about the length of 6 letters? what would be printed out?

Questions?

Exercise

Write a script that asks the user to guess a number between 1 and 10

- Assign a number to number_to_guess. Then ask the user for a number.
- Check to see if the input is number_to_guess equal to guess and if not ask for another input.
- if the input is equal to number_to_guess, then print a message "You Win!" and exit the loop.

That's all for now!

See you tomorrow!