Python Language

Enes Kemal Ergin

Questions of the Week

- How can I control my code?
- How to choose between 2 or more decisions?
- How many times can I run the same code?

Decision Making

- Decisions are taking place with If-Elif-Else statements.
- If statements are working with comparison statements, meaning that returns Boolean results. If the statements are True(1) decision will happened, if False(0), won't be happened.
- Python if statement selects actions to perform

```
>>> if 1:
        print('true')
>>> if not 1:
        print('true')
    else:
        print('false')
```

Decision Making(cont'd)

- You can select from multiway selections:

```
>>> x = 10
if x > 15:
    print("bigger than 15")
elif x < 5:
    print("smaller than 5")
else:
    print("in between 15 and 5 ")
```

Decision Making(cont'd)

- If statement may contain other statements, including other ifs. Called nested decision making

```
if (One >= 1) and (One <= 10):
    if (Two >= 1) and (Two <= 10):
        print("Your secret number is: ", One * Two)
    else:
        print("Incorrect second value!")
else:
    print("Incorrect first value!")
```

For those of you who're familiar with programming

- No switch statement…
- But if-elif statement does the same thing as switch statement.

Hands on time!

Iteration

- Iterative control statement is a control statement providing the repeated execution of a set of instructions(, looping)
- There are two ways of repeating execution in Python; using for and while loop.
- While loops are called indefinite loops
- For loops are called definite loops
- Before we going into looping we need to see very important concept which called variable updating.

Updating Variables

- A common pattern in assignment statements is an assignment statement that updates a variable - where the new value of the variable depends on the old.

```
x = x + 1
```

- But to do that first we need to **initialize** a x

```
x = 0
x = x + 1 ## or  x += 1
```

- Updating a variable by adding 1 is called an **increment**;
- subtracting 1 is called a **decrement**.

The while Statement

- One form of iteration in Python is the while statement. Here is a simple program that counts down from five and then says "Blastoff!".

```
n = 5
while n > 0:
        print(n)
        n = n-1
print ('Blastoff!‘)
```

- As I mentioned indentation is very important!
- While n is greater than 0 do the following statements.
  - Print the number
  - Decrement n by 1
- Because we are decrementing n by one in each iteration after 5 iteration loop will stop and we will see Blastoff!

The while Statement(Cont’d)

- In the example above, we have decrementation, which makes the loop definite.
- With out upgrading variables we cannot make our loop definite, because while loop is indefinite in its nature.

```
n = 5
while n > 0:
        print(n)
        # n = n-1
print('Blastoff!‘)
```

- This loop will not stop until you somehow interrupt the interpreter.

Infinite Loops and break

- Infinite loops are exists because there is no **iteration variable** telling them how many times to execute the loop.
- We can write an infinite loop on purpose and then use the break statement to jump out of the loop, for fun…

```
n = 10
while True:
        print (n, end = “ “)
        n = n - 1
print('Done!‘)
```

- This is definitely a infinite loop because logical expression in while statement is always True no matter what.
- Ctrl – C is a key combination to interrupt infinite loops, but you may choose the old version close the interpreter with pushing the red button.s

Infinite Loops and break(Cont'd)

- When a loop reached a break statement in loop, then it automatically exits the loop, no matter what.

```
while True:
        line = input('> ')
        if line == 'done':
                break
        print(line)
print('Done!‘)
```

- This will keep asking you to put input and show it to you, until you put done in the input part. Then it will reach the break and exits the loop.

Infinite Loops and continue

- If you would like to jump into another iteration while one is not finished completely, you can use continue to do that.
- Continue does not exit the loop it just passes the some part of the iteration.

```
while True:
        line = input('> ')
        if line[0] == '#' :
                continue
        if line == 'done':
                break
        print(line)
print('Done!‘)
```

- This code just skips the lines if you print #, which is a sign for comments in Python

The for Statement

- Sometimes we want to loop through a **set** of things such as a list of words, the lines in a file or a list of numbers.
- When we have a list of things to loop through, we can construct a *definite* loop using a for statement.

```
friends = ['Joseph', 'Glenn', 'Sally']
for i in friends:
        print('Happy New Year:', i)
print ('Done!‘)
```

- This examples uses a list data type but don’t worry about it we will learn next week.
- "Run the statements in the body of the for loop once for each i *in* the set named friends.“
- i is the iteration variable for the for loop, i changes for each iteration until items in friends ended.

Hands on Time!

To Do List

- Complete exercise #2 ~ Conditionals
- Complete exercise #2 ~ Iterations
- Read and go through Programming concepts-Week 3
- You didn’t got your project idea yet?