ENGL 516X: Methods of Formal Linguistic Analysis

Semester: Spring '18

Instructor: Sowmya Vajjala

Iowa State University, USA

30 January 2018

Class outline

- Week 3 Recap
- Loops in Python
- Loops: Practise exercise

Week 3 Recap

Topics discussed

- Boolean and logical expressions
- Conditional statements
- Exceptions and handling them
- Writing your own functions

Recap questions

- What is the difference between try/Except and if/else?

Recap questions

- What is the difference between try/Except and if/else?
- If I ask you to write a program that randomly prints numbers between 1 to 100, how will you approach it?

Recap questions

- What is the difference between try/Except and if/else?
- If I ask you to write a program that randomly prints numbers between 1 to 100, how will you approach it?
- Why do you think we need to write functions?

Recap questions

- What is the difference between try/Except and if/else?
- If I ask you to write a program that randomly prints numbers between 1 to 100, how will you approach it?
- Why do you think we need to write functions?
- What does a return statement do? Why do we need it?

Recap questions

- What is the difference between try/Except and if/else?
- If I ask you to write a program that randomly prints numbers between 1 to 100, how will you approach it?
- Why do you think we need to write functions?
- What does a return statement do? Why do we need it?
- What is the difference between a return statement and a print statement?

Terminology

- Function definition: The process of defining/creating a function
- Function parameters: variables inside the function definition, that can be used as input variables.
- Function call: The process of calling a function that was defined before.
- Function arguments: The variables or values that we pass as input to a function.
- Function object: The "data type" of a function call.
- Return value: Value returned by the function.
- Void function: A function that does not return anything.

Terminology: Examples

1. Consider the following:

   ```
   def example_print_function(some_string):
      print(some_string)
   ```

   This is a function definition. some_string is a parameter. This function does not return anything. So, it is a void function.

Terminology: Examples

1. Consider the following:

   ```
   def example_print_function(some_string):
      print(some_string)
   ```

   This is a function definition. some_string is a parameter. This function does not return anything. So, it is a void function.
2. Consider this statement: maximum = max(1,2,3,4,5)
   This is a function call. 1,2,3,4,5 are the arguments. max(1,2,3,4,5) is a function object. maximum is the return value.

Some rules for using functions in your code

- A function needs to be always first defined, before being called.
- A function definition starts with a def keyword.
- A function need not necessary have a return statement, but where you can use return instead of print, use it.
- Indentation and syntax needs to be strictly followed even with functions.

Solution to last class' exercise

Question: Expand the ctof, ftoc functions program to add the following conditions

- If the user chooses C and then enters a temparature beyond the range: [-58, +58] Celsius, print a message asking them to enter something realistic and stop. Else, do the conversion.
- If the user chooses F and then enters a temparature beyond the range: [-130, +130] Fahrenheit, print a message asking them to enter something realistic and stop. Else, do the conversion.

Solution: ThursdayHW.py in Module:Week4 on Canvas

Loops in Python

Iterations and loops: What and Why?

- What is a loop?

Iterations and loops: What and Why?

- What is a loop?
- What is iteration?

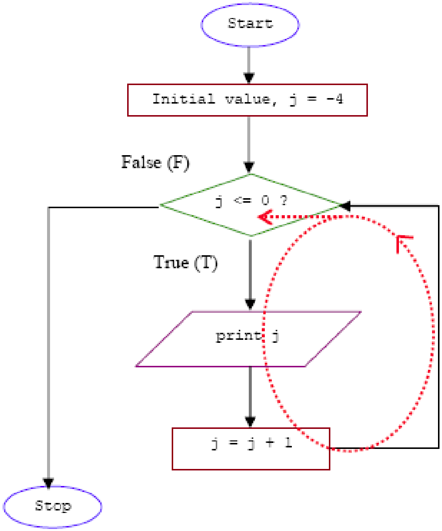
Iterations and loops: What and Why?

- What is a loop?
- What is iteration?
- Where are these concepts useful?

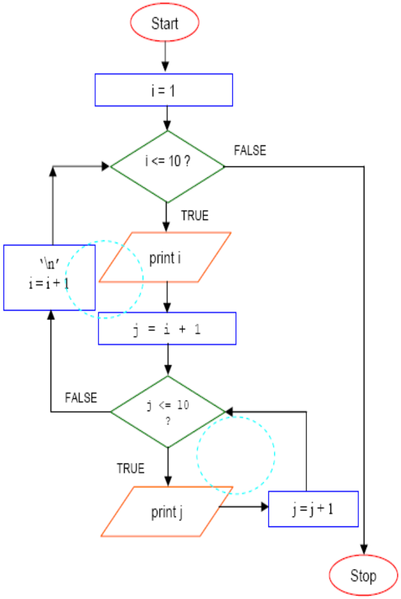
Iterations and loops: What and Why?

- What is a loop?
- What is iteration?
- Where are these concepts useful?

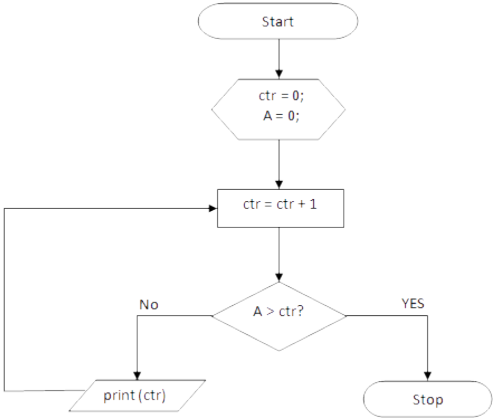
An example loop flowchart



A flowchart with a loop in a loop



A flowchart with an infinite loop



Loops in Python: a "while" Loop

```
i=0
while i<3:
    print(i)
    i = i+1
print("Done with the while loop!")
```

Loops in Python: a ”for” Loop

```
i=0
for i in range(0,3):
    print(i)
    i=i+1
print("Done with the for loop!")
```

(There is also another way of looping using a ”for”. We will get to that next week.

Example Program with While loop

```
def someFunction(number):
    result = 1
    i = 1
    while i<=number:
        result = result*i
        i = i+1
    return result
print(someFunction(5))
```

Same program with For loop

```
def someFunction(number):
    result = 1
    for i in range(1,number+1):
        result = result*i
    return result
print(someFunction(5))
```

Infinite Loops

- Example 1:

```
tempstring = "whatever"
while tempstring == "whatever":
    print(tempstring)
print("You will never see this message")
```

- Example 2:

```
while True:
   print("I won't stop!")
print("You will never see this message")
```

- these kind of loops will never stop until you apply force (on your keyboard, that is).

Even such patterns can be put to use!

break statement in Python

Suppose you want to keep taking input from the user until the user enters "done"

```
while True:
    line = input('Enter something: ')
    if line == 'done':
        print("Stopping here")
        break #break statement breaks the loop.
    else:
        print(line)
print('Done!')
```

continue statement in python

Break lets you out of the loop completely. continue just comes out of current iteration and goes to the next iteration of the loop.

```
while True:
    line = input("Enter something: ")
    if line == 'pass':
       print("I am passing without printing what you entered")
       continue
    elif line == 'done':
       print("I am stopping the program.")
       break
    else:
       print(line)
print("Done!")
```

Choosing between a for and while

- Use a for loop if you know, before you start looping, the maximum number of times that you?ll need to execute the body.
- if you are required to repeat some computation until some condition is met, and you cannot calculate in advance when (of if) this will happen, use while.
- Anything implemented in for, can have a while counterpart and vice versa.
- My preference: for over while (because we wont get into infinite loops by mistake)

Exercise: Spot the bug

```
def someFunction(number):
    result = 1
    i = 1
    while i<=number:
        result = result*i
        number = number+1
    return result
print(someFunction(5))
```

Exercise: analyze this

```
def seq3np1(n):
    while n != 1:
        print(n, end=", ")
        if n % 2 == 0:          # n is even
            n = n // 2
        else:                   # n is odd
            n = n * 3 + 1
    print(n, end=".\n")
```

What will seq3np1(16) print?

Exercise: analyze this

```
def seq3np1(n):
    while n != 1:
        print(n, end=", ")
        if n % 2 == 0:           # n is even
            n = n // 2
        else:                    # n is odd
            n = n * 3 + 1
    print(n, end=".\n")
```

What will seq3np1(16) print? 16, 8, 4, 2, 1

Programming Exercise

- Ask the user to enter a number first (integer). Assign it to a variable n.
- Now, take input from the user n number of times after this. These have to be numbers.
- Once the input taking is done, you have print the following back to the user: sum of the these numbers, and average.
- Example interaction with your program:

```
> Enter the number of numbers you want to enter:
5
> Enter a number: 2
> Enter a number: 6
> Enter a number: 5
> Enter a number: 3
> Enter a number: 8
> The sum of these numbers is: 24
> The average of these numbers is: 4.8
```

- Assume for now that the user is following your directions, and there are no errors to handle.

Solution Discussion

(Solution is uploaded after the class as: LoopQuestion.py)

Extending this program

Add exception handling to this program, to address the following conditions:

- n is a integer between 2 and 100
- Each subsequent number is a integer in the range 0 to 10000
- If the user enters a string or floating point or any non-integer for n, print an error message using try and except and stop.
- If the user enters anything other than a number after n, detect their mistake using if and else, and print an error message and move on to take next input number.
- Note: This is Similar to Final exercise in Chapter 5 in the textbook
- Note 2: I am not asking you to organize this program into functions - but think if you can.

Additional Exercise for fast programmers

- Write a program that takes a number and prints multiplication table for that number (n*1 to n*10, one number per line)
- Expected input/output:

```
> Enter a number: 5
> 5*1 = 5
  5*2 = 10
  5*3 = 15
  .. ...
  5*10 = 50
```

Next Class

- Topics: Revision of what we learnt so far
- writing a main() function - good programming practices
- ToDo: In the forum post for Revision topics - post the topics you want me to discuss on Thursday. I will accomodate as many requests as possible.
- There will also be in-class programming exercises as usual on thursday.
- If you want little bit more challenging stuff, look at Chapter 7 in the second textbook: `http://openbookproject.net/thinkcs/python/english3e/iteration.html`