ENGL 516X:
Methods of Formal Linguistic Analysis

Semester: Spring '18

Instructor: Sowmya Vajjala

Iowa State University, USA

25 Jan 2018

Class outline

- Last class' exercise
- Built-in functions in Python
- Functions vs Methods
- Errors, Debugging and Exception Handling
- Writing our own python functions
- Summing it up exercise

Last class' question

One of the solutions from Discussion Forum

```
***F->C***
inp = input('Enter Fahrenheit Temperature: ')
fahr = float(inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)
***C->F***
inp = input('Enter Celsius Temperature: ')
cel = float(inp)
fahr = (cel*9/5)+32
print(fahr)
```

Built-in functions in Python

Some functionalities are already implemented in Python, we don't need to write our code. Some examples are:

- print()
- type()
- int(), float(), str()
- min(), max()
- input()
- ord()

Built-in functions: Small exercise

Go to the list of built-in functions in python.
(`https://docs.python.org/3/library/functions.html`)
Pick five functions we did not discuss in the class, whose purpose you can guess going by the name. See the function description and your expectation match.

Python Modules

- modules are large .py files that implement several functions.
- a collection of such modules is called a python package
- if we want to use a module in our program, we type:
  import module_name
  at the top of the program (you can type in the middle too, if the functions from this module are called after that line, but normally, putting all imports on top is the convention)
- e.g., random module - has functions to generate random numbers in different sequences (useful in scientific computing)
- e.g., math module - has mathematical functions (e.g., logarithm, square root etc)
- all modules:
  `https://docs.python.org/3/py-modindex.html`

functions vs methods

- We use len("python") but "python".isalpha() - what is the difference?
- The first one is called a "function", second one is a "method" that works for strings.

functions vs methods

- We use len("python") but "python".isalpha() - what is the difference?
- The first one is called a "function", second one is a "method" that works for strings.
- simple difference: functions - may work with several kinds of data types (e.g., print() works with integers, strings, floats, lists etc).
- methods are tied to specific data objects (i.e., .isalpha() works only for string variables.

functions vs methods

- We use len("python") but "python".isalpha() - what is the difference?
- The first one is called a "function", second one is a "method" that works for strings.
- simple difference: functions - may work with several kinds of data types (e.g., print() works with integers, strings, floats, lists etc).
- methods are tied to specific data objects (i.e., .isalpha() works only for string variables.
- complex difference: There is something called "object oriented programming" - which is beyond the scope of this class.

Writing our own functions

- Why?: To implement our own custom sequence of programming events.
- Advantage: reusability (I call print() so many times - I don't write code for print() functionality each time!)

Writing our own functions

- Why?: To implement our own custom sequence of programming events.
- Advantage: reusability (I call print() so many times - I don't write code for print() functionality each time!)
- Functions can sometimes take arguments
  ⇒ e.g, if you use len() function to get the length of a string, you need to give it something in those brackets. len() throws an error. len("Python") shows something.

Writing our own functions

- Why?: To implement our own custom sequence of programming events.
- Advantage: reusability (I call print() so many times - I don't write code for print() functionality each time!)
- Functions can sometimes take arguments
  ⇒ e.g, if you use len() function to get the length of a string, you need to give it something in those brackets. len() throws an error. len("Python") shows something.
- Some functions have optional arguments (print() and print("something") - it won't throw an error.

Writing our own functions

- Why?: To implement our own custom sequence of programming events.
- Advantage: reusability (I call print() so many times - I don't write code for print() functionality each time!)
- Functions can sometimes take arguments
  $\Rightarrow$ e.g, if you use len() function to get the length of a string, you need to give it something in those brackets. len() throws an error. len("Python") shows something.
- Some functions have optional arguments (print() and print("something") - it won't throw an error.
- some functions don't take any arguments (e.g., locals() -returns currently assigned variables, and other internal python function names etc.

lyrics.py example from textbook

```
def print_lyrics():
    print("I’m a lumberjack, and I’m okay.")
    print('I sleep all night and I work all day.')

def repeat_lyrics():
    print_lyrics() \#-this is a function "call"
    print_lyrics() \#-this is a function "call"

repeat_lyrics() \#-this is a function "call"
```

- What is the sequence of statements in this? What gets executed first?

Understanding the flow of execution

- Functions have to be defined before they are "called".
- Program execution starts from the first statement of a program (or, if there is a "Main" function defined. More on this later)
- When Python sees a function call, it takes a detour, goes to that function, runs through that, and "returns" to where it stopped earlier.

Arguments and Parameters

```
def add_str_int(some_string, some_int):
   print(some_string, str(some_int))
#some_string, some_int are parameters.

add_str_int("python",3) #"python", 3 - are arguments.
add_str_int("Week", 4) #"Week", 4 - are arguments
```

Fruitful function and void function - 1

Fruitful functions - give you back something as output, which you assign to some variable in your program.

```
def sum_two_numbers(a,b):
   return a+b
#Note: the above a, b - are only within that function.
#they don't exist beyond that.
a = sum_two_numbers(5,6)
b = a**3
print(b)
```

Fruitful function and void function - 2

Void functions do not "return" anything.

```
def sum_two_numbers(a,b):
    a = a+1
    b = b+1
    print(a + b)

c = sum_two_numbers(2,3)
print(c)
```

Errors and Handling Them in code

- How it looks if you do not write some error handling code in your program:

```
Enter a number: <user enters "Sowmya">
<output looks like:>
Traceback (most recent call last):
... ...
ValueError: invalid literal for int(): "Sowmya"
```

- How it looks if you write some code to handle these kind of errors:

```
Enter a number: <user enters "Sowmya">
<output looks like:>
Please enter a valid number
```

Errors and Handling Them in code

In Python, we use a kind of conditional statement called try,except to do error handling. It is like an if-else, but serves a different purpose - used when we expect possible errors that may occur (e.g., division by 0, wrong input etc)

```
try:
   number = input("Enter a number: ")
   next_number = int(number)+1
   print(int(number)/next_number)
except Exception as e:
   print(e)
```

Note the indentation.

Error Handling: Another Example

This one can perhaps be handled by series of if else statements?

```
try:
   number = int(input("Enter a integer number between 1 and 100: "))
   if number >=1 and number <101:
     print("you entered: " + str(number))
   else:
     print("You entered a number, but not between 1 and 100")
except:
   print("Please enter a valid number, and betwen 1 and 100")
```

Read this program description:

Read this program description:

- It has two functions: ctof, ftoc - celsius to F, F to celsius (what you wrote on Tuesday)
- Prompt the user to choose a temparature scale: C or F. If something else is chosen, you should stop the program there, with a message suggesting them to type either C or F.
- If the user entered C, call CtoF, print the output.
- If they chose F, call FtoC, print the output
- I will provide starter code for this, which handles possible exceptions.
- Rule for this exercise: You should not use print() statements within these function definitions.

Extra exercise:

Extend this program to do this:

- 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.

Next Week

- Topics: Writing loops, Commenting your code; Revision so far.
- Readings for the week: Chapter 5 in textbook
- Optional practice problems: Uploaded on Canvas.
- Other practice problems: at the end of Chapter 4 in the textbook