Introduction to Application Development in Python

Lecture 3

Fatih Han Çağlayan

Questions? Previous lecture topics

- Overview and structure of final assignment
- What to expect when learning to program?
- Install the python environment and IDE
  - Latest Python 3
  - IDE Preference: PyCharm
- Modules
- Integrated Development Environment (IDE)
- Functions
- Loops and main loop of program
- Conditions
- Output and input

- Warm-up assignment with debugger
- Implement main loop, version command, help command and quit command

Lecture topics

- String operations
- List operations
- For loop in python
- Typecasting

String operations

- Capitalize — **str.capitalize**

  'abcd'.capitalize() => 'Abcd'

- Count — **str.count**

  'aabbcda'.count('a') => 3

- Find ('oo' in string) — **str.find**

  'foo'.find('oo') => 1

- Is numeric? — **str.isnumeric**

  '123'.isnumeric() => **True**

- Is alphabetic? — **str.isalpha**

  '123'.isalpha() => **False**

- Split (on the space character) — **str.split**

  'a and b and c and d'.split(' and ') => ['a', 'b', 'c', 'd']

- String presence ('great') — **in** keyword

  'great' **in** 'my code is great' => **True**

Input notation calculator

- Prefix: sum 1 2
- Postfix: 1 2 add
- Infix: 1 add 2

- Prefix notation is mandatory
- The remaining other two are advanced requirements

- Make use of 40 in total built-in string operations Python 3

```
tokens = 'sum 1 2'.split()
```

yields

```
['sum', '1', '2']
```

```
function_name = tokens[0]
operands = tokens[1:]
function_name in known_functions
```

Access specific members of list

List operations

- Append (append the number 15) — **list.append**

  [1, 2, 3, 4].append(15) => [1, 2, 3, 4, 15]

- Remove (remove the number 15) — **list.remove**

  [1, 2, 3, 4, 15].remove(15) => [1, 2, 3, 4]

- Count — **list.count**

  len([1, 2, 3, 4, 15]) => 5

- Reverse — **list.reverse**

  [1, 2, 3, 4, 15].reverse() => [15, 4, 3, 2, 1]

- Sort — **list.sort**

  [15, 3, 4, 2, 1].sort() => [1, 2, 3, 4, 15]

Passing list into function

- Suppose we have the following function:

```
def sum(a, b):
        return a+b
```

- Two ways of passing a list into this function:

```
operands = [3, 8]
sum(operands[0], operands[1])

sum(*operands)
```

- For the first case, the number of operands is always fixed
- The latter is the better option, because it will work in any case

Type casting

- Suppose we have the following function:

```
def sum(a, b):
        return a+b
```

- This will not work because the members of the list are strings

```
operands = ['3', '8']
sum(operands[0], operands[1])
```

- After this operation it will work, this is called type casting

```
operands[0] = int(operands[0])
operands[1] = int(operands[1])
```

Type casting

- Type casting is not always possible. This will give an error

```
int('1a')
```

  - Solution 1: check in advance if conversion is possible:

```
operand = '1a'
if operand.isnumeric():
    operand_as_integer = int(operand)
```

  - Solution 2: handle the actual error:

```
operand = '1a'
try:
    operand_as_integer = int(operand)
except (ValueError):
    ...
```

- Python has conversion functions for many types (**bool**, **float**, **int**, **list**, **str**, etcetera)

For loops in python

- Traditionally used when you have a block of code which you want to repeat a fixed number of times.
- Python for statement iterates over the members of a sequence in order, executing the block each time.
- Compare with "while" loop, used when a condition needs to be checked each iteration or to loop forever

```
operands = [1, 2, 3, 4, 5]
for operand in operands:
        print(operand)

for x in range(0, 3):
        print("We're on time %d times!" % (x))

exit = False
while (not exit):
        do_something()
        exit = should_exit()
```

- What is the output of the last statement?

Assignment

- Please see schedule and assignments on course website:
  - Implement expression processing

- Deliver assignment **both** in person and automated testing