

Web Server Design

Lecture 2 – Introduction to Python

Old Dominion University

Department of Computer Science

CS 431/531 Spring 2026

David Calano <dcalao05@odu.edu>

What is Python?

- A free and open source programming language
- Scripting language
- Interpreted and Compiled
- Cross-platform
- Dynamically typed
- Object-oriented (but not enforced)
- White-spaces for block indentation
- Integrates with other languages
- Developed in 1980s

Why Python?

- Fast development and prototyping
- Easy to test
- Rich standard library
- Rich community contributed libraries and modules
- Less boilerplate code
- Easy to read and write

Expression vs. Statement

Expression

- Represents something
- Python **Evaluates** it
- Results in a value
- Example:
 - "Hello" + " " + "World!"
 - $(5*3)-1.4$

Statement

- Does something
- Python **Executes** it
- Results in an action
- Example:
 - `print("Hello World!")`
 - `import os`

Differences with C/C++ Syntax

- White spaces for indentation
- No "{}" for blocks
- Blocks begin with ":" (in the preceding line)
- No type declarations needed
- No ++, -- operators
- Several differences in keywords
- No && and || ("and" and "or" instead)

Interactive Python Shell

```
$ python
Python 3.7.4 (default, Sep 12 2019, 15:40:15)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
        file:  a file-like object (stream); defaults to the current sys.stdout.
        sep:   string inserted between values, default a space.
        end:   string appended after the last value, default a newline.
        flush: whether to forcibly flush the stream.

>>> print("Hello World!")
Hello World!
>>> 2 + 3 * 4 / 5
4.4
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>> exit()
$
```

Simple Data Types

- **Integer:** 7
- **Float:** 87.23
- **String:** "abc", 'abc'
- **Bytes:** b"abc", b'abc'
- **Boolean:** False, True

Compound Data Types

- **List:** ["Hello", "There"]
- **Tuple:** ("John", "Doe", 35)
- **Set:** {"Python", "Ruby", "Perl"}
- **Dictionary:** {"name": "John Doe", "age": 35}

String

- Concatenation: "Python" + "Rocks" → "PythonRocks"
- Repetition: "Python" * 2 → "PythonPython"
- Slicing: "Python"[2:3] → "th"
- Size: len("Python") → 6
- Index: "Python"[2] → 't'
- Search: "x" in "Python" → False
- Comparison: "Python" < "ZOO" → True (lexicographically)

String vs. Bytes

- Bytes object is a sequence of bytes
- String is a sequence of characters (often in UTF-8)

```
>>> type("Hello")
<class 'str'>
>>> type(b"Hello")
<class 'bytes'>
>>> b"Hello".decode()
'Hello'
>>> "Hello".encode()
b'Hello'
```

List

- Equivalent to arrays
- `X = [0, 1, 2, 3, 4]`
 - Creates a pre-populated array of size 5.
- `Y = []`
 - Creates an empty list
- `X.append(5)`
 - X becomes [0, 1, 2, 3, 4, 5]
- `len(X)`
 - Gets the length of X which is 6

List

```
>>> mylist = [0, 'a', "hello", 1, 2, ['b', 'c', 'd']]  
>>> mylist[1]  
a  
>>> mylist[5][1]  
c  
>>> mylist[1:3]  
['a', "hello", 1]  
>>> mylist[:2]  
[0, 'a', "hello"]  
>>> mylist[3:]  
[1, 2, ['b', 'c', 'd']]  
>>> mylist.remove('a')  
>>> mylist  
[0, "hello", 1, 2, ['b', 'c', 'd']]
```

List

- >>> `mylist.reverse()` → Reverse elements in list
- >>> `mylist.append(x)` → Add element to end of list
- >>> `mylist.sort()` → Sort elements in list ascending
- >>> `mylist.index('a')` → Find first occurrence of 'a'
- >>> `mylist.pop()` → Removes last element in list

Tuple

- `X = (0, 1, 2, 3, 4)`
 - Creates a pre-populated list of **fixed** size 5
 - Immutable (can't be changed)
- `print(X[3]) #=> 3`

List vs. Tuple

- Lists are mutable, tuples are immutable
- Lists can be resized, tuples can't
- Tuples can be faster than lists

Dictionary

- An array indexed by strings (equivalent to hashes)

```
>>> marks = {"science": 90, "art": 25}  
>>> print(marks["art"])  
25  
>>> marks["chemistry"] = 75  
>>> print(marks.keys())  
["science", "art", "chemistry"]
```

Dictionary

- `dict = { "fish": 12, "cat": 7}`
- `'dog' in dict` (Is 'dog' a key?)
- `dict.keys()` (Gets a list of all keys)
- `dict.values()` (Gets a list of all values)
- `dict.items()` (Gets a list of all key-value tuples)
- `dict["fish"] = 14` (Assignment)

Variables

- Everything is an object
- No need to declare
- No need to assign
- Not strongly typed
- Assignment = reference
 - Ex: `>>> X = ['a', 'b', 'c']
>>> Y = X
>>> Y.append('d')
>>> print(X)
['a', 'b', 'c', 'd']`

User Input

Without a Message:

```
>>> x = input()
```

```
3
```

```
>>> x
```

```
'3'
```

With a Message:

```
>>> x = input('Enter the number: ')
```

```
Enter the number: 3
```

```
>>> x
```

```
'3'
```

Evaluate User Input

```
>>> x = input()
```

```
3+4
```

```
>>> x
```

```
'3+4'
```

```
>>> eval(x)
```

```
7
```

File Read

```
>>> f = open("input_file.txt", "r")
```

File handle Name of the file Mode

```
>>> line = f.readline()
```

Read one line at a time

```
>>> f.close()
```

Stop using this file and close

File Write

```
>>> f = open("output_file.txt", "w")
```

File handle

Name of the file

Mode

```
>>> line = f.write("Hello World!")
```

Write a string to the file

```
>>> f.close()
```

Stop using this file and close

Control Flow

- Conditions:
 - if
 - if / else
 - if / elif / else
- Loops:
 - while
 - for
 - for loop on iterators

Conditions

- The condition must be terminated with a colon ":"
- Scope of the loop is the following indented section

```
>>> if score == 100:  
    print("You scored a hundred!")  
elif score > 80:  
    print("You are an awesome student!")  
else:  
    print("Go and study!")
```

While Loop

```
>>> i = 0  
>>> while i < 10:  
    print(i)  
    i = i + 1
```

Do not forget the  at the end of the condition!

For Loop

```
>>> for i in range(10):  
    print(i)
```

```
>>> myList = ['hany', 'john', 'smith', 'aly', 'max']  
>>> for name in myList:  
    print(name)
```

Do not forget the **:** at the end of the condition!

Inside vs. Outside Block

```
for i in range(3):  
    print("Iteration {}".format(i))  
print("Done!")
```

Iteration 0

Done!

Iteration 1

Done!

Iteration 2

Done!

```
for i in range(3):  
    print("Iteration {}".format(i))  
print("Done!")
```

Iteration 0

Iteration 1

Iteration 2

Done!

Loop Over a File Iterator

```
>>> f = open ("my_file.txt", "r")
>>> for line in f:
    print(line)
```

Pass Empty Block

- It means do nothing

```
>>> if x > 80:  
    pass  
else:  
    print("You are less than 80!")
```

Break the Loop

- It means **quit** the loop

```
>>> for name in myList:  
    if name == "aly":  
        break  
    else:  
        print(name)
```

→This will print all names before “aly”

Continue to the Next Iteration

- It means skip this iteration of the loop

```
>>> for name in myList:  
     if name == "aly":  
         continue  
     else:  
         print(name)
```

→This will print all names except “aly”

Functions

- Finding the biggest number in a list:

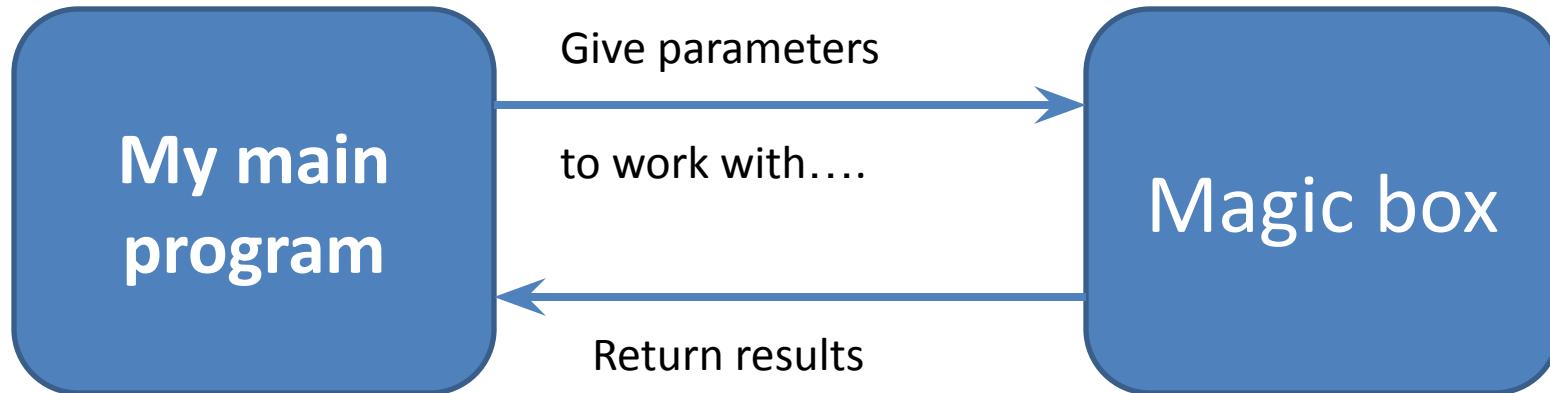
```
mylist = [2,5,3,7,1,8,12,4]
maxnum = 0
for num in mylist:
    if (num>maxnum):
        maxnum = num
print("The biggest number is: {}".format(maxnum))
```

Functions

- What if the code is a bit more complicated and long?
- What if the same logic is repeated?
- Writing the code as one blob is bad!
 - Harder to read and comprehend
 - Harder to debug
 - Rigid
 - Non-reusable

Functions

```
def my_funtion(parameters):  
    do stuff
```



Functions

- Back to our example:

```
mylist = [2,5,3,7,1,8,12,4]
maxnum = getMaxNumber(mylist)
print("The biggest number is: {}".format(maxnum))
```

Functions

- Implement the function getMaxNumber as you wish

```
def getMaxNumber(list_x):  
    maxnum = 0  
    for num in list_x:  
        if (num > maxnum):  
            maxnum = num  
    return maxnum
```

Testing

```
def getMaxNumber(list_x):
```

```
    """
```

```
        Returns the maximum number from the supplied list
```

```
>>> getMaxNumber([4, 7, 2, 5])
```

```
7
```

```
>>> getMaxNumber([-3, 9, 2])
```

```
9
```

```
>>> getMaxNumber([-3, -7, -1])
```

```
-1
```

```
"""
```

```
maxnum = 0
```

```
for num in list_x:
```

```
    if (num>maxnum):
```

```
        maxnum = num
```

```
return maxnum
```

```
if __name__ == '__main__':
```

```
    import doctest
```

```
doctest.testmod()
```

```
$ python max_num.py
```

```
*****
```

```
File "max_num.py", line 8, in __main__.getMaxNumber
```

```
Failed example:
```

```
    getMaxNumber([-3, -7, -1])
```

```
Expected:
```

```
    -1
```

```
Got:
```

```
    0
```

```
*****
```

```
1 items had failures:
```

```
    1 of  3 in __main__.getMaxNumber
```

```
***Test Failed*** 1 failures.
```

Functions

```
def getMaxNumber(list_x):  
    return max(list_x)
```

Functions

- All arguments are passed by value
- All variables are local unless specified as global
- Functions in Python can have several arguments or None
- Functions in Python can return several results or None

Functions Return Multiple Values

```
def getMaxNumberAndIndex(list_x):
    maxnum = 0
    index = -1
    i = 0
    for num in list_x :
        if (num>maxnum):
            maxnum = num
            index = i
        i = i + 1
    return maxnum, index
```

Calling Multi-values Function

```
mylist = [2,5,3,7,1,8,12,4]
maxnum, idx = getMaxNumberAndIndex(mylist)
print("The biggest number is: {}".format(maxnum))
print("It's index is: {}".format(idx))
```

Class

```
class Student:  
    count = 0                      # class variable  
    def __init__(self, name):       # Initializer  
        self.name = name  
        self.grade = None  
        Student.count += 1  
    def updateGrade(self, grade):   # Instance method  
        self.grade = grade  
  
if __name__ == "__main__":          # Execute only if script  
    s = Student("John Doe")  
    s.updateGrade("A+")  
    s.grade                         #=> "A+"  
    Student.count                   #=> 1
```

Comments

```
mylist = [2,5,3,7,1,8,12,4]
# The function getMaxNumberAndIndex will be called next to retrieve
# the biggest number in list "mylist" and the index of that number
maxnum, idx = getMaxNumberAndIndex(mylist)
print("The biggest number is: {}".format(maxnum))
print "It's index is: {}".format(idx))
```

Python Files

- Python files end with ".py"
- To execute a python file you write:

```
$ python myprogram.py
```

Python Scripts

- To make the file “a script”, set the file permission to be executable and add this shebang in the beginning:

```
#!/usr/bin/python
```

or better yet

```
#!/usr/bin/env python3
```

Modules

- We just call the math library that has the perfect implementation of square root

```
>>> import math  
>>> x = math.sqrt(9.0)
```

Or

```
>>> from math import sqrt  
>>> x = sqrt(9.0)
```

Command-line Arguments

- To get the command line arguments:
- *>>> import sys*
- The arguments are in *sys.argv* as a *list*

Exception Handling

```
>>> sum_grades = 300
>>> number_of_students = input()
0
>>> average = sum_grades / number_of_students
→ Error! Divide by Zero
```

Remember: User input is evil!

Exception Handling

```
try:  
    average = sum_grades / number_of_students  
except:  
    # This catches if something wrong happens  
    print("Something wrong happened, please check it!")  
    average = 0
```

Exception Handling

```
try:  
    average = sum_grades / number_of_students  
except ZeroDivisionError:  
    # This catches if something wrong happens  
    print("Something wrong happened, please check it!")  
    average = 0
```

Exception Handling

```
try:  
    number_of_students = input()  
    average = sum_grades / number_of_students  
except ZeroDivisionError:  
    # This catches if a number was divided by zero  
    print("You Evil User!.....you inserted a zero!")  
    average = 0  
except IOError:  
    # This catches errors happening in the input process  
    print("Something went wrong with how you enter words")  
    average = 0
```

Generators

```
>>> def fib():
        a = b = 1
        while True:
            yield a
            a, b = b, a + b

>>> f = fib()
>>> print(next(f))    #=> 1
>>> print(next(f))    #=> 1
>>> print(next(f))    #=> 2
>>> print(next(f))    #=> 3
>>> print(next(f))    #=> 5
```

Regular Expression

- A powerful tool to deal with patterns in String and Bytes

```
>>> import re
>>> status = "HEAD /foo HTTP/1.1"
>>> pattern = re.compile(r"^( [A-Z]+ )\s+([S+])\s+([A-Z0-9/\. ]+)$")
>>> m = pattern.match(status)
>>> m.groups()
('HEAD', '/foo', 'HTTP/1.1')
```

<https://docs.python.org/3/howto/regex.html>