Web Science:
Intro to Python
(Part 1 - Overview and Basics)

CS 432/532

Old Dominion University

Permission has been granted to use these slides from Hany SalahEldeen, Sawood Alam, and Alexander Nwala, Michele C. Weigle

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 (no {})
- 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

Differences with C/C++ Syntax

- White spaces for block indentation, no {}

- No { } for blocks
  - blocks begin with : (in the preceding line)

- No type declarations needed

- No ++, -- operators

- Several differences in keywords
  - and, or instead of && , ||

- No switch/case

Expression vs. Statement

Expression

- *Represents* something
- Python *evaluates* it
- Results in a *value*

- Examples:

```
"Hello" + " " + "World!"

(5*3)-1.4
```

Statement

- *Does* something
- Python *executes* it
- Results in an *action*

- Examples:

```
print("Hello World!")

import os
```

Rule of Thumb: If you can print it or assign it to a variable, it's an expression. If you can't, then it's a statement.
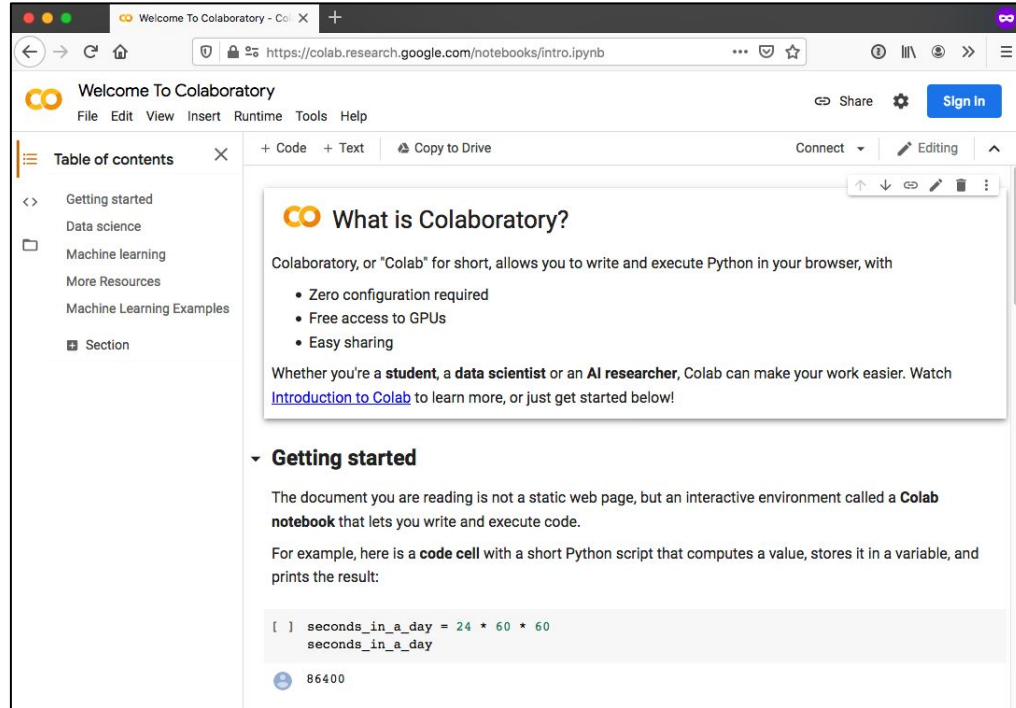
Accessing the Python Interpreter

- Run locally on your own computer
  - download and install from Welcome to Python.org

- Login to ODU-CS Linux server
  - `ssh linux.cs.odu.edu`
  - or, use PuTTY to login to linux.cs.odu.edu (see Dr. Zeil's CS 252 notes, "Logging In")

- Google Colab Notebooks

Google Colab Notebooks

Ref: Google Colab notebooks

Code examples for the rest of the slide set are also in a Google Colab notebook

Interactive Python Shell

```
$ python3
Python 3.6.9 (default, Apr 18 2020, 01:56:04)
[GCC 8.4.0] 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)
[...]
>>> print("Hello World!")
Hello World!
>>> 2 + 3 * 4 / 5
4.4
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>> exit()
$
```

On CS Linux machines, use **python3** instead of python

Make sure you're using Python 3, not Python 2

Simple Data Types

- **Integer:**   7
- **Float:**   87.23

- **Boolean:**   `False`, `True`

- **String:**   `"abc"`, `'abc'`
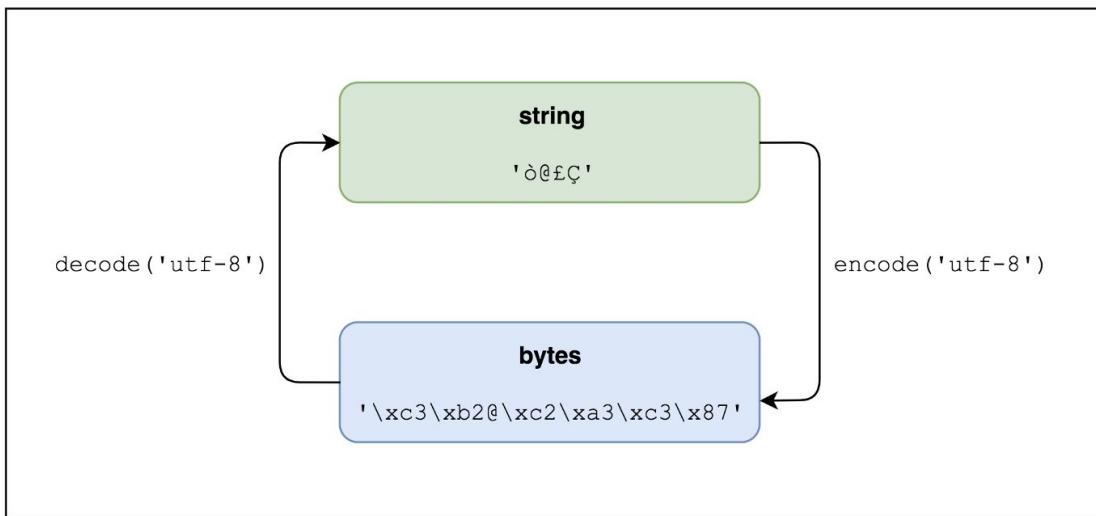- **Bytes:**   `b"abc"`, `b'abc'`

String vs. Bytes

- Bytes object is a sequence of bytes
- String is a sequence of characters that must be encoded (often in Unicode, UTF-8)

```
>>> type("Hello")
<class 'str'>

>>> type(b"Hello")
<class 'bytes'>
```

`bytes` and `str` instances can't be used together with operators like `>` and `+`

Ref: Strings, Unicode, and Bytes in Python 3: Everything You Always Wanted to Know

Decoding Bytes, Encoding Strings


```
string

'ò@£Ç'
```

```
decode('utf-8')
```

```
encode('utf-8')
```

```
bytes

'\xc3\xb2@\xc2\xa3\xc3\x87'
```

```
>>> b"Hello".decode()
'Hello'

>>> "Hello".encode()
b'Hello'
```

Ref: Strings, Unicode, and Bytes in Python 3: Everything You Always Wanted to Know

Manipulating Strings

"Unicode
Sandwich"

"Bytes on the outside, unicode on the inside, encode/decode at the edges."

bytes	*Input*
decode()	
str	*Manipulation*
encode()	
bytes	*Output*

Ref: [Strings, Unicode, and Bytes in Python 3: Everything You Always Wanted to Know](#)
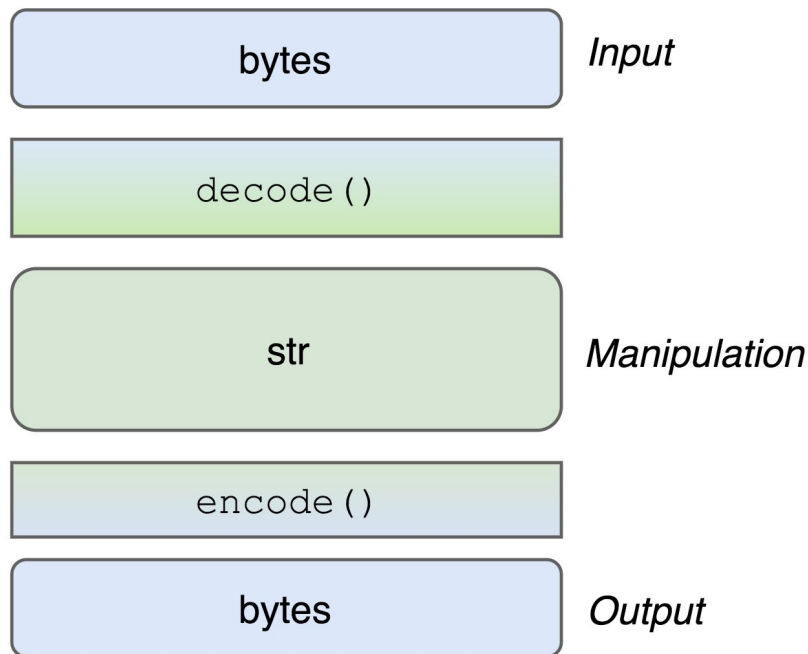
String

- Concatenation: `"Python"+"Rocks"` → `"PythonRocks"`
- Repetition: `"Python" * 2`   → `"PythonPython"`
- Size: `len("Python")`   → `6`
- Index: `"Python"[2]`   → `'t'`
- Slicing: `"Python"[2:4]`   → `"th"`
                     `"Python"[:4]`   → `"Pyth"`
- Search: `"th" in "Python"` → `True`
- Comparison: `"Python" < "ZOO"` → `True`
  (lexicographically)

Compound Data Types

- **List:**   `["Hello", "There"]`

- **Tuple:**   `("John", "Doe", 35)`

- **Set:**   `{"Python", "Ruby", "Perl"}`

- **Dictionary:**   `{"name": "John Doe", "age": 35}`

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)`
  - Returns the length of X, which is 6

List

```
>>> mylist.reverse()  → Reverse elements in list

>>> mylist.append(x)  → Add element to end of list

>>> mylist.sort()     → Sort elements in list
                         ascending order

>>> mylist.index('a') → Find first occurrence of 'a'

>>> mylist.pop()      → Removes last element in list
```

Ref: Python documentation, "More on Lists"

List

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

```
>>> unsorted = [32, 18, 17, 2, 5]
>>> unsorted.sort()
>>> unsorted
[2, 5, 17, 18, 32]
```

Tuple

```
>>> X = (0, 1, 'a', 4, 3)
```
- Creates a pre-populated list of fixed size 5

```
>>> print(X[3])
4

>>> X[0] = 'b'
```

Lists vs. Tuples
- Lists are mutable, tuples are immutable (can't be changed)
- Lists can be resized, tuples can't
- Tuples can be faster than lists

```
----------------------------------------------------------------
TypeError                         Traceback (most recent call last)
<ipython-input-51-24d6d5b10aa9> in <module>()
----> 1 X[0] = 'b'

TypeError: 'tuple' object does not support item assignment
```

Ref: Python documentation, "Tuples and Sequences"

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 key-value tuples*

- `dict["fish"] = 14` *Assignment*

Ref: Python documentation, "Dictionaries"

Variables

- Everything is an object

- Assignment = reference

- No need to declare

- No need to assign

- Not strongly typed

```
>>> X = ['a', 'b', 'c']
>>> Y = X
>>> Y.append('d')
>>> print(X)
['a', 'b', 'c', 'd']
```

X → │ 'a' │ 'b' │ 'c' │ 'd' │

Y ↗
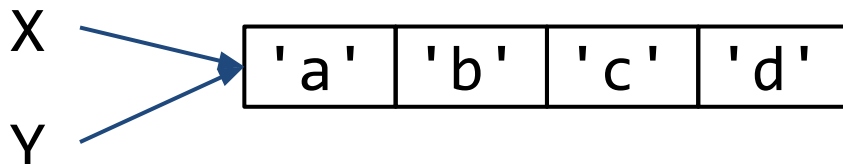
Comments

```python
# This creates and populates a list
mylist = [2,5,3,7,1,8,12,4]

# This prints out Hello World!
print("Hello World!")
```

Web Science:

Intro to Python

(Part 2 - I/O and Conditionals)

CS 432/532

Old Dominion University

Permission has been granted to use these slides from Hany SalahEldeen, Sawood Alam, and Alexander Nwala, Michele C. Weigle

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

Ref: [Python documentation, "Reading and Writing Files"](#)

Manipulating Files

- `readline()` - reads a line from file

- `readlines()` - reads all the file as a list of lines

- `read()` - reads all the file as one string

Loop Over a File Iterator

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

>>> for line in f:
        print(line)
```

File Write

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

File handle   Name of the file   Mode

```
>>> 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

Conditional

- 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)
```

range(*start=0, stop, step=1*) - generates integer numbers between the given start and stop

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

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

in - can be used in a loop to iterate through a list or in a conditional to test if a specific value is in a list

not in - can be used in a conditional to test if a specific value is *not* in a list

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!
```

Note the print statement here. It uses {} to denote where the result
should appear and the `str.format()` function to format the variable.

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

```
>>> myList = ['hany','john','smith','aly','max']
>>> 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

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

→This will print all names except "aly"

Web Science:

Intro to Python

(Part 3 - Functions and Modules)

CS 432/532

Old Dominion University

Permission has been granted to use these slides from Hany SalahEldeen, Sawood Alam, and Alexander Nwala, Michele C. Weigle

Find the Largest Number in a List

```python
mylist = [2,5,3,7,1,8,12,4]
maxnum = 0
for num in mylist:
    if (num>maxnum):
        maxnum = num
print("The largest number is {}".format(maxnum))
```
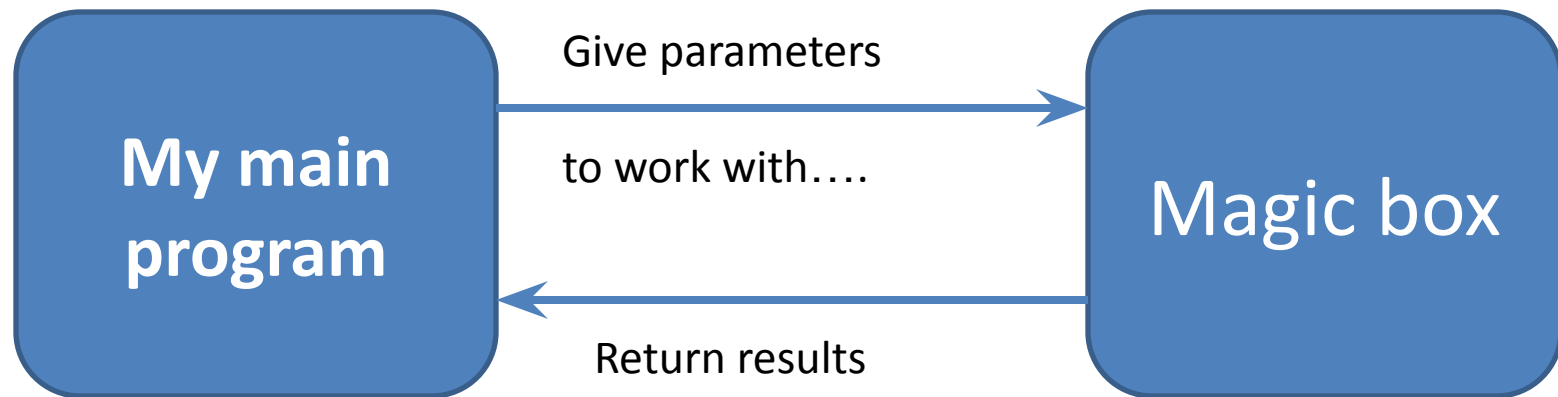
Note the print statement here. It uses {} to denote where the result should appear and the `str.format()` function to format the variable.

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_function(parameters):
do stuff

My main program → Give parameters to work with…. → Magic box

Magic box → Return results → My main program

Back to the example...

```
mylist = [2,5,3,7,1,8,12,4]

maxnum = getMaxNumber(mylist)

print("The largest number is {}".format(maxnum))
```

Functions

- Implement the function getMaxNumber as you wish

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

Is Anything Wrong with getMaxNumber()?

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


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


>>> getMaxNumber([-3, -7, -1])
0
```

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

New getMaxNumber()

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

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

For this example, there turns out to already be a built-in function to find the largest number in a list (max). getMaxNumber isn't really needed.

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 Can Return Multiple Values

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

Calling a Multi-Value Function

```
mylist = [2,5,3,7,1,8,12,4]
maxnum, idx = getMaxNumberAndIndex(mylist)
print("The largest number is {} and its index is {}" .
format(maxnum, idx))
```

```
The largest number is 12 and its index is 6
```

Note the print statement here. Now we have two {} instances. The variables are separated by a comma in the `str.format()` function.

Modules

- Python contains lots of modules with useful functions.

- Use the `import` statement to access the functions available in a module

Ref: [Python Module Index](#)

Module Example

```
>>> import math
>>> x = math.sqrt(9.0)
```
Or
```
>>> from math import sqrt
>>> x = sqrt(9.0)
```
Or
```
>>> from math import sqrt as sq
>>> x = sq(25.0)
```

Web Science:
Intro to Python
(Part 4 - Running Python, Error Handling)
CS 432/532

Old Dominion University

Permission has been granted to use these slides from Hany SalahEldeen, Sawood Alam, and Alexander Nwala, Michele C. Weigle

Python Files

Python files end with `.py`

To execute a Python file named `myprogram.py`

    `$ python myprogram.py`

On ODU-CS Linux machines, use `python3` instead of `python`
(use Python version 3, not version 2)

    `$ python3 myprogram.py`

*Remainder of the slides will assume we're
on ODU-CS Linux and will use python3*

```
$ python --version
Python 2.7.17

$ python3 --version
Python 3.6.9
```

Python Scripts

- Scripts can be run without using the `python3` command

①
```
#!/usr/bin/python3
# myprogram.py

print ("Hello World!")
```

```
$ which python3
/usr/bin/python3
```

②
```
$ chmod a+x myprogram.py
$ ls -l myprogram.py
-rwxr-xr-x 1 mweigle proxy 58 Jun  4 11:21 myprogram.py*
```

③
```
$ ./myprogram.py
Hello World!
```

Command-Line Arguments

- To access command line arguments:
import sys

```
$ ./testargs.py weigle 432 532
```

- The arguments are in sys.argv as a list

  `len(sys.argv)` - returns the number of arguments

  `sys.argv[0]` - holds the name of the script

Ref: [sys — System-specific parameters and functions — Python 3.8.5 documentation](#)

Command-Line Arguments

```python
#!/usr/bin/python3
# testargs.py

import sys

print ("{} is the name of the script." . format(sys.argv[0]))
print ("There are {} arguments: {}" . format(len(sys.argv), str(sys.argv)))

for ind, arg in enumerate(sys.argv):
    print ("[{}]: {} {}".format(ind, arg, sys.argv[ind]))
```

```
$ ./testargs.py weigle 432 532
./testargs.py is the name of the script.
There are 4 arguments: ['./testargs.py', 'weigle', '432', '532']
[0]: ./testargs.py ./testargs.py
[1]: weigle weigle
[2]: 432 432
[3]: 532 532
```

Error Handling

What happens when you have an error?

```
>>> sum_grades = 300
>>> num_students = input()
>>> average = sum_grades / int(num_students)
```

What if the user entered 0?

Error Handling

What happens when you have an error?

```
>>> sum_grades = 300
>>> num_students = input()
0
>>> average = sum_grades / int(num_students)
    ZeroDivisionError: division by zero
```

Exception Handling

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

Exception Handling

```python
try:
 average = sum_grades / int(num_students)
except ZeroDivisionError:
 # This catches if a number was divided by zero
 print("You tried to divide by zero!")
 average = 0
```

Exception Handling

```python
try:
 num_students = input()
 average = sum_grades / int(num_students)
except ZeroDivisionError:
 # This catches if a number was divided by zero
 print("You tried to divide by zero!")
 average = 0
except IOError:
 # This catches errors happening in the input process
 print("Something went wrong with how you enter words")
 average = 0
```

Automated Testing with `doctest`

```python
# max_num.py

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()
```

```
$ python3 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.
```

Ref: [doctest — Test interactive Python examples — Python 3.8.5 documentation](#)

Web Science:

Intro to Python

(Part 5 - Regular Expressions and Web Libraries)

CS 432/532

Old Dominion University

Permission has been granted to use these slides from Hany SalahEldeen, Sawood Alam, and Alexander Nwala, Michele C. Weigle

Regular Expressions

- Access functions with `import re`
- Describe the regular expression pattern with `compile()`
- Apply the pattern starting at the beginning of a string with `match()`

- [ ] specifies a *character class*, range of characters to match
  - [abc] matches any of the characters 'a', 'b', or 'c'
  - [^abc] matches any character **NOT** 'a', 'b', or 'c'

- Shortcuts to particular classes
  - \d   [0-9]   \D   [^0-9]
  - \s   [\t\n\r\f\v] (any whitespace)   \S (non-whitespace)
  - \w   [a-zA-Z0-9_] (any alphanumeric) \W (non-alphanumeric)

Ref: Regular Expression HOWTO — Python 3.8.5 documentation

Regular Expressions

- To repeat patterns
  - * matches 0 or more times
  - \+ matches 1 or more times

- To specify position in the line
  - ^ requires start of the line
  - $ requires end of the line

- Divide matches into groups with ( )

Ref: [Regular Expression HOWTO — Python 3.8.5 documentation](#)

Regular Expression Example

`HEAD` `/foo` `HTTP/1.1`

`^([A-Z]+)\s+(\S+)\s+([A-Z0-9\/\.]+)$`

`^([A-Z]+)`	1 or more capital letters at the beginning of line
`\s+`	1 or more whitespace character
`(\S+)`	1 or more non-whitespace character
`\s+`	1 or more whitespace character
`([A-Z0-9\/\.]+)$`	1 or more letters, numbers, `/`, or `.` at the end of the line

Regular Expressions

```
>>> import re

>>> req = "HEAD /foo HTTP/1.1"

>>> pattern =
 re.compile(r"^([A-Z]+)\s+(\S+)\s+([A-Z0-9\/\.]+)$")

>>> m = pattern.match(req)

>>> m.groups()
('HEAD', '/foo', 'HTTP/1.1')
```

match() returns None if no match is found

groups() returns a tuple containing all of the subgroups

Online regex tester: https://regex101.com/

Python Libraries: requests

The de facto standard for making HTTP requests in Python

```
import(requests)
response = requests.get('http://example.com')
```

Response headers and body are all stored in `response`

Refs: Requests: HTTP for Humans™ — Requests 2.24.0 documentation
Python's Requests Library (Guide) – Real Python

requests: Response Headers

```python
import requests
response = requests.get('http://example.com')

print ("Status Code: {}".format(response.status_code))
print ("URI: {}\n".format(response.url))


print ("Headers: {}\n".format(response.headers))
print ("Date: {}".format(response.headers['Date']))
print ("Content-Type: {}".format(response.headers['Content-Type']))
print ("Content-Length: {}".format(response.headers['Content-Length']))
```

Refs: Python's Requests Library (Guide) – Real Python
redirection: Requests documentation, "Redirection and History"

requests: Request Parameters

```python
import requests
response = requests.get('http://google.com/search', params={'q': 'LSU'})

print ("URI requested: {}".format(response.request.url))
print ("Status Code: {}\n".format(response.status_code))

# split the string into a list, one line per element
lines = response.text.splitlines()

# use loop to only print first 5 lines of the response
for i in range(5):
 print(lines[i])
```

Ref: Python's Requests Library (Guide) – Real Python

Beautiful Soup: HTML/XML Parser

```python
from bs4 import BeautifulSoup
import requests


response = requests.get('http://google.com/search', params={'q': 'LSU'})


soup = BeautifulSoup(response.text)
for links in soup.find_all('a'):
  print(links.get('href'))
```

Refs: [Beautiful Soup Documentation — Beautiful Soup 4.9.0 documentation](#)
[Beautiful Soup 4 Python, PythonForBeginners](#)

Objectives

- Explain the differences between Python and C/C++ syntax.
- Execute a simple Python on the ODU-CS Linux server and in a Google Colab notebook.
- Describe the differences between a tuple, list, and dictionary.
- Write a Python program that accepts command-line arguments.
- Write a Python program that uses the requests library to access a webpage.
- Write a Python program that uses the BeautifulSoup library to extract all of the links in a webpage.