A Python Primer

Dr. Yuankai Qi

Faculty of SET / School of Computer Science

part source from Rolf Schwitter@ Macquarie University

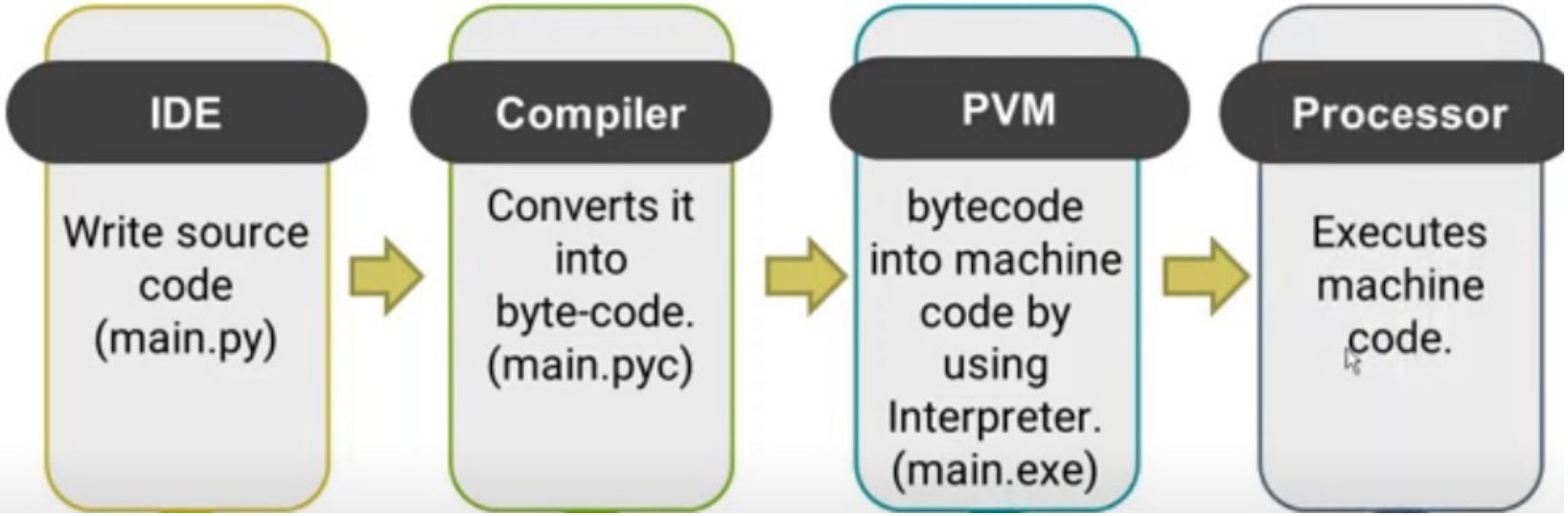
Why do we learn programming languages?

To let computers do what we want to!

Why Python?

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is designed to be **highly readable**.
- Python has good **frameworks and libraries**.
- Python is good for **fast prototyping**.
  - ✓ Much faster than Java/C++ to test a new hypothesis.

How is python code be conducted?



Outcomes

1. Implement the simple IR method.

- **Approach in traditional IR:**

Document:

Star Wars: Episode VII
Three decades after the defeat of the Galactic Empire, a new threat arises.

Query:
star wars the force awakens reviews

$$q = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \xrightarrow{f(q,d)} d = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

$$f_{VSM}(q,d) = \frac{\langle q,d \rangle}{\| q \| \cdot \| d \|}$$

- Representing query and document as word vectors

Outcomes

2. Given a mark, outputs the grade.

Mark	Grade	
85-100%	High Distinction	HD
75-84%	Distinction	D
65-74%	Credit	C
50-64%	Pass	P
0-49%	Fail	F
0%	Fail No Submission	FNS
	Result Pending	RP

Outlines

- Variables
- Sequences
- Dictionaries
- Loops
- Functions
- Modules
- Files I/O
- Classes

Python: Shell (Anaconda)

```
(base) qyk@qyk ~ % python
Python 3.7.6 (default, Jan  8 2020, 13:42:34)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

https://www.anaconda.com/products/distribution

https://www.python.org/downloads/

Variables

- Variables in Python follow the standard nomenclature of **an alphanumeric name** *beginning in a letter or underscore*.
- Variable names are **case-sensitive**.
- Variables do not need to be declared.
- All variables are **objects.**
- Data types of variables are **inferred from the assignment** statement.

```
>>> abc = 0   >>> Abc = 0   >>> abc0 = 0   >>> oabc = 0
>>> _abc = 0  >>> a_b_c = 0  >>> abc_= 0   >>> a?bc = 0

>>> myFirstVar = 0   >>> my_First_Var = 0
```

Variables

Types

```
>>> a = 4                # Integer
>>> a
4
>>> b = 3.2              # Float
>>> b
3.2
>>> c = "TECH1004"       # String
>>> c
'TECH1004'
>>> d = True             # Boolean

>>> a = "5"              # Rebound to string
>>> a
'5'
```

Sequences

- The most basic data structure in Python is the sequence.
- The most common sequences in Python are:
  - strings
  - lists
  - tuples.
- **Each element** of a sequence is assigned to **an index**.
- **The first index is zero**.

Strings

```
>>> name = "Catherine Ringier"
>>> name
'Catherine Ringier'
>>> quotes = "Rock 'n' Roll"    # quotation marks
>>> quotes
"Rock 'n' Roll"
>>> long_string = """This string consists of
three different lines and ends with a
full stop."""
>>> long_string
'This string consists of\nthree different lines and ends
with a\nfull stop.'
```

Accessing Values in a String

```
>>> name = "Catherine Ringier"
>>> name[0]          # Slice
'C'
>>> name[1]
'a'
>>> name[-1]
'r'
>>> name[0:9]        # Range slice
'Catherine'
>>> name[:9]
'Catherine'
>>> name[-7:]
'Ringier'
```

Additional String Operators

```
>>> first_name = "Catherine"
>>> second_name = "Ringier"
>>> name = "Catherine" + " " + "Ringier"    # Concatenation
>>> name
'Catherine Ringier'
>>> "t" in first_name                       # Membership
True
>>> "z" not in first_name
True
>> s = "bla"
>>> s*3                                     # Repetition
'blablabla'
```

String Formatting Operator

```
>>> name = "Bob"
>>> text = f"Hello, {name}"
>>> text
'Hello, Bob'
```

String Escaping

```
>>> ?
She said: "I don't care".
```

```
>>> print('She said: "I don\'t care".')
She said: "I don't care".
>>> print('''She said: "I don't care".''')
She said: "I don't care".
>>> print("""She said: "I don't care".""")
She said: "I don't care".
>>>
```

String Escaping

```
\\              # Backslash
\'              # Single quote
\"              # Double quote
\b              # ASCII Backspace
\n              # Newline
\t              # Tab
```

String Methods

```
>>> s = "Whereof one cannot speak,  thereof one must be silent."
>>> s.upper()
'WHEREOF ONE CANNOT SPEAK, THEREOF  ONE MUST BE SILENT.'
>>> s.lower()
'whereof one cannot speak, thereof  one must be silent.'
>>> s.count("o")
5
>>> s.find("one")
8
>>> s.replace("silent", "noisy")
'Whereof one cannot speak, thereof one must be noisy.'
```

String Methods

```
>>> string = "Hello TECH1004!"
>>> string.startswith("Hello")
True
>>> string.endswith("TECH1004!")
True
>>> string.index("o")
4
>>> string.split(" ")
['Hello', 'TECH1004!']
```

Conditionals

Goal: Input mark, output grade

Mark	Grade	
85-100%	High Distinction	HD
75-84%	Distinction	D
65-74%	Credit	C
50-64%	Pass	P
0-49%	Fail	F
0%	Fail No Submission	FNS
	Result Pending	RP

Conditionals

```
if condition1:
    # do …
elif condition2:
    # do …
```

indent

.

.

```
else: # optional
    # do …
```

write conditionals for grading marks

```
mark = 66

if mark >= 85:
    print("HD")
elif mark >= 75:
    print("D")
elif mark >= 65:
    print("Cr")
elif mark >= 50:
    print("P")
else:
    print("F")
```

Conditionals

Indent

Example

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

Example

Syntax Error:

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

Boolean Operators

Multiple conditions

or: at least one condition is met

and: all conditions are met

not: opposite, it always outputs True or False

```
>>> x = 5
>>> x < -4 or x > 4
True
>>> if x > 4 and x < 6:
        print("five")

five
>>> not x
False
```

Conditionals

What if we want to test multiple grades?

```
mark = 66

if mark >= 85:
    print("HD")
elif mark >= 75:
    print("D")
elif mark >= 65:
    print("Cr")
elif mark >= 50:
    print("P")
else:
    print("F")

mark = 77

if mark >= 85:
    print("HD")
elif mark >= 75:
    print("D")
elif mark >= 65:
    print("Cr")
elif mark >= 50:
    print("P")
else:
    print("F")
```

Functions

If a piece of code is used frequently, it should be written in the form of functions.

A function is defined using the def keyword.

A function only run when it is called.

A function can return data as a result.

```
>>> def my_func():
        print("Hello from a function")
>>> my_func()
Hello from a function
>>> def add_2(num): # parameter(s)
        """ return 2  more than num  """
        return num + 2
```

Functions

```
>>> def add_n(num, n=3):    # default parameters
    """n defaults to 3"""
       return num + n

>>> add_n(2)
5
>>> res = add_n(15, 5)
>>> res
20
>>>
```

The variable s is an object

What is “object”?

```
>>> s = "Whereof one cannot speak,  thereof one must be silent."
>>> s.upper()
```

Class

Object is an instance of a class.

Class is a higher-level organization of variables and functions. To create a class, use the keyword class:

```
>>> class Dog():
        def __init__(self, name, age):
            self.name = name
            self.age = age
        def getName():
            print(self.name)
>>> dog = Dog("Luca",3 )
>>> dog.getName( )
Luca
```

Classes

```
class Shape ():

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.x * self.y

    def perimeter(self):
        return 2 * self.x + 2 * self.y

# Note: "__init__" is a special method in Python classes, it is the
constructor method for a class. "self" is a reference to the class instance.
```

Classes

```
def   describe(self,   text):
    self.description = text

def   authorName(self, text):
    self.author = text

def   scaleSize(self, scale):
    self.x  =  self.x  *  scale
    self.y = self.y * scale
```

Using Classes

```
rectangle = Shape(100, 45)

# finding the area of your rectangle:
print(rectangle.area())

# finding the perimeter of your rectangle:
print(rectangle.perimeter())

# describing the rectangle
rectangle.describe("A wide rectangle, more than twice\
 as wide as it is tall")

# making the rectangle 50% smaller
rectangle.scaleSize(0.5)

# re-printing the new area of the rectangle
print(rectangle.area())
```

Lists

- A list is a sequence of Python objects to store multiple pieces of information.
- Use **square brackets** to define lists

```
>>> list1 = ['COMP225', 'COMP249', 'COMP329', 'COMP348']
>>> list2 = [1, 2, 3, 4, 5, 6]
>>> list3 = ["a", "b", "c", "d"]
>>> list4 = [1,”a”,2.5]
```

Accessing Lists

```
>>> list1 = ['COMP225', 'COMP249', 'COMP329', 'COMP348']
>>> list2 = [1, 2, 3, 4, 5, 6]
>>> list3 = ["a", "b", "c", "d"]
>>> list4 = [1,”a”,2.5]


>>> list1[0]
'COMP225'
>>> list2[-1]
6
>>> list3[2:4]
['c', 'd']
```

Updating Lists

```
>>> list1=['COMP225', 'COMP249', 'COMP329', 'COMP348']

>>> list1[0] = 'COMP202'
>>> list1
['COMP202', 'COMP249', 'COMP329', 'COMP348']

>>> list1.append('COMP355')
>>> list1
['COMP202', 'COMP249', 'COMP329', 'COMP348', 'COMP355']
```

Delete List Elements

```
>>> del list1[1:3]
>>> list1
['COMP202', 'COMP348', 'COMP355']
```

List Operations

```
>>> len(list1)                                  # Length
3
>>> list2 + list3                               # Concatenation
[1, 2, 3, 4, 5, 6, 'a', 'b', 'c', 'd']
>>> 5 in list2                                  # Membership
True
>>> list1[1:]                                   # Slicing
['COMP348', 'COMP355']
>>> for x in [1, 2, 3]:                         # Iteration
        print(x)
1
2
3
```

List Methods

```
>>> a = []
>>> a.append(5)
>>> a.append(3)
>>> a.append(1)
>>> sorted(a)       # sorted works on all iterables[1]
[1, 3, 5]
>>> a
[5, 3, 1]
>>> a.sort()        # sorts a list in-place; works only on lists
>>> a
[1, 3, 5]
```

[1]Sequence (string, tuple, list) or collection (set, dictionary, frozen set) or any iterator.

List Methods

```
>>> a.reverse()
>>> a
[5, 3, 1]
>>> a.remove(5)    # remove an element
>>> a
[3, 1]
>>> a.pop()
1
>>> a
[3,]
```

Tuples

- A tuple is a sequence of immutable Python objects.
  - ✓ That means in contrast to lists, tuples cannot be changed.
- Tuples usually use **parentheses**, whereas lists use square brackets.
- However, note it's the **comma** not the parentheses that define a tuple.
- The following is the same:

```
>>> tupleA = "a", "b"
>>> tupleB = ("a", "b")
>>> tupleA == tupleB
True
```

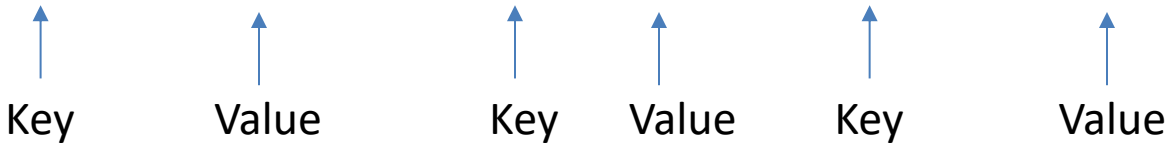
Tuples

```
>>> tuple1 = ('COMP225', 'COMP249', 'COMP329', 'COMP348')
>>> tuple2 = (1, 2, 3, 4, 5, 6)
>>> tuple3 = ("a", "b", "c", "d")
>>> tuple1[2]
'COMP329'
>>> tuple2[2]
3
>>> tuple3[2:3]
('c',)          # Note: trailing comma in one element tuple.
>>> tuple3[2:4]
('c', 'd')
>>> tuple4 = ("e",)
>>> tuple3 + tuple4
('a', 'b', 'c', 'd', 'e')
```

Dictionaries

- A dictionary consists of unordered key-value pairs. Each key is separated from its value by a **colon** (`:`).
- The items are separated by **commas**, and a set of key-value pairs is enclosed in **curly braces**.

```
>>> dict = {'name':'John', 'age':23, 'pref':[3, 5, 8]}
```



- The values of a dictionary can be of any Python data type.
- An empty dictionary looks as follows: `{}`.

Dictionaries

```
>>> dict = {'name':'John', 'age':23, 'pref':[3, 5, 8]}
>>> dict['name']
'John'
>>> dict['name'] = "Max"
>>> dict
{'name': 'Max', 'age': 23, 'pref': [3, 5, 8]}
>>> del dict['age']
>>> dict
{'name': 'Max', 'pref': [3, 5, 8]}
>>> list(dict.keys())
['name', 'pref']
>>>
```

Dictionaries

```
>>> 'name' in dict
True
>> dict.items()
dict_items([('name', 'Max'), ('pref', [3, 5, 8])])
>>> dict.keys()
dict_keys(['name', 'pref'])
>>> dict.update({'age':34})
>>> dict
{'name': 'Max', 'pref': [3, 5, 8], 'age': 34}
>>> dict.values()
dict_values(['Max', [3, 5, 8], 34])
```

for Loop

```
>>> for number in [1, 2, 3]:
        print(number)
1
2
3

>>> for number in range(1, 1000):
        print(number)
1
2
3
```

for Loop

```
>>> units = ["COMP249", "COMP329", "COMP348"]
>>> for info in units:
        print(info)
COMP249
COMP329
COMP348
>>> for index, value in enumerate(units):
        print(index, value)
0 COMP249
1 COMP329
2 COMP348
>>> for item in enumerate(units):
        print(item)
(0, 'COMP249')
(1, 'COMP329')
(2, 'COMP348')
```

for Loop

```
>>> dict = {'name':'John', 'age':23, 'pref':[3, 5, 8]}
>>> for k, v in dict.items():
      print(f"{k}:{v}")
```

while Loop

```
units = ["COMP249", "COMP329", "COMP348"]
idx = 0
while idx < 3:
    print(idx, units[idx])
    idx += idx
```

```
0 COMP249
1 COMP329
2 COMP348
```

while Loop

```
while True:
    n = input("Please enter 'exit' to quit: ")
    if n.strip() == 'exit':
        break
```

Modules

- A module allows you to logically organize your Python code.
- Simply, a module is a file consisting of Python code.

Example

Save this code in a file named `mymodule.py`

```
def greeting(name):
  print("Hello, " + name)
```

Import the module named mymodule, and call the greeting function:

```
import mymodule

mymodule.greeting("Jonathan")
```

Modules

- A module allows you to logically organize your Python code.
- Simply, a module is a file consisting of Python code.

```
# Import module math
>>> import math

# Now you can call a defined function in that module
# as follows:
>>> math.sqrt(3)
1.732050807568772
```

Modules

- from ***X import a, b, c*** imports the module ***X***, and creates references in the current namespace to the given objects.
- You can now use ***a*** and ***b*** and ***c*** in your program.

```
>>> from math import factorial, pi, sqrt
>>> factorial(3)
6
>>> pi
3.141592653589793
>>> sqrt(3)
1.732050807568772
```

Modules

How to know what functions are in a module?

The dir() Function

```
>>> import math
>>> names = dir(math)
>>> names
['__doc__', '__loader__', '__name__', '__package__',
'__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',
'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf',
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

dir() looks at all the properties / attributes and methods of that object

Text File: unit-description.txt

- Approach in traditional IR:

Document:

Star Wars: Episode VII
Three decades after the defeat of the Galactic Empire, a new threat arises.

Query:
star wars the force awakens reviews

$$q = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \xrightarrow{f(q,d)} d = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

$$f_{VSM}(q,d) = \frac{\langle q,d \rangle}{\| q \| \cdot \| d \|}$$

- Representing query and document as word vectors

Text File: unit-description.txt

Artificial Intelligence (AI) is a well-established field that studies how computers and computer software capable of exhibiting intelligent behaviour can be designed. In this unit students will be exposed to fundamental concepts in AI such as agent architecture, knowledge representation, planning and search, as well as their application in some topical domains. Upon completion of this unit students will be able to apply problem-solving strategies that are required to build intelligent systems.

Reading Files

```
# Read lines
file = open("unit-description.txt", "r")
print(file.readlines())
file.close()

# Read entire text without line breaks
with open("unit-description.txt", "r") as file:
    print(file.read())

# Loop over lines
file = open("unit-description.txt", "r")
for line in file:
    print(line)
file.close()
```

Writing Files

```
file = open("unit-description.txt", "w")
file.write("This is a test.\n")
file.write("To add more lines.")
file.close()

file = open("unit-description.txt", "a")
file.write("This is for append.")
file.close()
```

Exceptions

- Syntax:

```
try:
      You do your operations here;
      ......................
except ExceptionI:
      If there is ExceptionI, then execute this block.
except ExceptionII:
      If there is ExceptionII, then execute this block.
      ......................
else:
      If there is no exception then execute this block.
```

Exceptions

```
try:
   file = open("test-file", "w")
   file.write("This is my test file for exception handling.")
except IOError:
   print "Error: can\'t find file or write data"
else:
   print("Successfully written content to test file.")
   file.close()
```

Features and Tricks

```
>>> # Unpacking
>>> a, b, c = [1, 2, 3]
>>> a, b, c
(1, 2, 3)
>>> # Generator
>>> [i for i in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> # Help
>>> help(str)
Help on class str in module builtins:

 class str(object)
  |  str(object='') -> str
  …
```

Features and Tricks

```
>>> # Flatten a list in one line: list comprehension
>>> list = [[1, 2, 3], [4, 5, 6], [7], [8, 9]]
>>> flat_list =  [y for x in list for y in x]
>>> flat_list
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> # Dictionary comprehension
>>> {x: x**3 for x in range(10)}
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343,
```

Take-Home Messages

- Python is an easy to learn programming language.
- Python is interpreted, interactive, and object-oriented.
- Python is well designed, fast, robust, portable, and scalable.
- These are important factors for AI applications.
- Python is good for prototyping; most AI is research-related.
- Python comes with a number of AI and ML libraries:
  https://wiki.python.org/moin/PythonForArtificialIntelligence
- Check Python documentation for details:
  https://docs.python.org/3/
- Python tutorial
  `https://www.w3schools.com/python/default.asp`