Computer Science 2

Lecture 7

Introduction to Python

Learning goals

- Identify the main similarities and differences between *Java* and *Python* (e.g. compiler vs interpreter, statically vs dynamically typed, etc.)
- Be able to implement solutions for *basic problems* using Python
- Understand and be able to utilize *variable types* not available in *Java* (or not covered in CS1-CS2)

The Python Environment

- **Python is Interpreted**
  - Python is processed at runtime by the interpreter. You do not need to compile your program before executing it
- **Python is Interactive**
  - You can actually sit at a Python prompt and interact with the interpreter directly to write your programs
- **Python is Object-Oriented**
  - Python supports the Object-Oriented style or technique of programming that encapsulates code within objects
  - But Python is really bad at it

Script mode programming

- Rather than begin execution at main, Python begins at the top of the file/script
- When it gets to the bottom, the interpreter stops
- Two versions of python
  - Python2 and Python3
  - Some differences in syntax
  - Python2 reached end of life (EOL) in 2020
- Python files have the extension **.py**
  - E.g., main.py
- Python execution
  - $ python main.py

Some Important Differences

- Indentation matters
- Python variables are not declared, nor do they have set types
- Python skips all those semicolons
- No interface type in Python
- Python is more concise
- Naming

Indentation matters

We generally write Java code like this because it is readable

Java

```
public class HelloWorld
{
    public static  void main(String[] args)
    {
            int variable = 0;
            for(int i=0; i<5; i++)
            {
                variable  = variable + i;
                System .out.println(variable);
            }
    }
}
```

Indentation matters

But the language does not care about the newlines or indentation

The following code runs just the same

Java

```
public class HelloWorld{public static void main(String[] args){int variable=0; for(int i=0; i<5; i++){ variable=variable+i;
System.out.println(variable);}}}
```

But Python cares very much

This code must look just like this in Python to work

```
variable = 0
for i in range(5):
    variable  = variable + i
    print(variable)
```

Variable typing

- In Java, every variable
  - Must be declared
  - With a type
  - And can hold values of only that type through its entire life
  - Known as nominative, static typing
- In Python, variables
  - Need not be declared
  - Are not assigned a type
  - Can hold values of any type throughout their lives
  - Known as duck, dynamic typing

Variable typing

- Java uses nominative typing
  - With nominative typing, a variable is of a given type if it is declared as such (or if a type's association with the variable is inferred through mechanisms such as object inheritance)
  - The type of a variable is what it is declared to be
- Python uses duck typing
  - With duck typing, an object is of a given type if it has all the methods and properties required by that type
  - If a variable supports a given operation, then it belongs to a type that supports that operation
  - Basically, try it and find out
  - If it looks like a duck, swims like a duck, and quacks like a duck, it is probably a duck

Variable typing

- Java uses static typing
  - With static typing, the type of a variable is known before the program runs
- Python uses dynamic typing
  - With dynamic typing the type of a variable is not known until the program is running

[Interlude: Duck typing

```
>>> var = 10
>>> print(var)
10
>>> var = “ten”
>>> print(var)
ten
>>> var = [1, 0]
>>> print(var)
[1, 0]
```

https://medium.com/swlh/duck-typing-in-python-3-practical-examples-a40d8113266c

Semicolons

- Java uses semicolons to separate statements
  - People use newlines to make the code readable
- Python uses newlines to separate statements
  - No semicolons required

Interfaces

- Java uses interfaces to support limited multiple inheritance
- Python just supports multiple inheritance directly

Conciseness

Java

```
public class HelloWorld
{
  public static void main( String[] args)
    {
      System.out.println(“hello world!”);
    }
}
```

Python

```
print(“hello world!”)
```

Naming

Java

Python

Summary: Python vs Java

Dimension	Java	Python
Verbosity	Verbose	Concise
Performance	Compiled, faster	Interpreted, slower
Learning curve	Easy	Easier than Java
Typing discipline	Static, nominative	Dynamic, duck
Popular in	Embedded and cross-platform applications	Data Science, AI, Machine Learning

Details

- We will take a look at some of the language details of Python to give a bit of flavor
- Some types that require classes in Java are built in to Python
- If statements and loops are similar, but use indentation
- Functions do not have return types or exception declarations
  - Parameters do not have types
  - Multiple values can be returned at once

Variable types

- Python has five standard types (in addition to classes)
  1. Numbers (int, long, float, etc.)
  2. Strings
  3. Lists
  4. Tuples
  5. Dictionaries

Numbers

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point

>>> var      =  10
>>> print(2 * var)
20
>>> var      =  10.0
>>> print(2 * var)
20.0
```

Strings

```
>>> str = “Hello World!”
>>> print(str)                  # Prints complete string
Hello World!
>>> print(str[0])               # Prints first character of the string
H
>>> print(str[2:5])             # Prints characters starting from 3rd to 5th
llo
>>> print(str[2:])              # Prints string starting from 3rd character
llo World!
>>> print(str * 2)              # Prints string two times
Hello World!Hello World!
>>> print(str + “TEST”)         # Prints concatenated string
Hello World!TEST
```

Lists

```
>>> list = [ “abcd”, 786 , 2.23, “john”, 70.2]
>>> tinylist = [123, “john”]
>>> print(list)                          # Prints complete list
[“abcd”, 786, 2.23, “john”, 70.2]
>>> print(list[0])                 # Prints first element of the list
abcd
>>> print(list[1:3])               # Prints elements starting from 2nd till 3rd
[786, 2.23]
>>> print(list[2:])                # Prints elements starting from 3rd element
[2.23, “john”, 70.2]
```

Lists

```
>>> print(list[-1])            # Prints the last element
[70.2]
>>> print(tinylist * 2)              # Prints list two times
[123, “john”, 123, “john”]
>>> print(list + tinylist)           # Prints concatenated lists
[ “abcd”, 786 , 2.23, “john”, 70.2, 123, “john”]
>>> tinylist.append(10)              # Append elements in the list
>>> print(tinylist)
[123, “john”, 10]
```

Tuples

```
>>> tuple = (“abcd”, 786 , 2.23, “john”, 70.2)
>>> tinytuple = (123, “john”)
>>> print(tuple)                    # Prints complete tuple
(“abcd”, 786 , 2.23, “john”, 70.2)
>>> print(tuple[0])                 # Prints first element of the tuple
abcd
>>> print(tuple[1:3])               # Prints elements starting from 2nd till 3rd
(786, 2.23)
>>> print(tuple[2:])                # Prints elements starting from 3rd element
(2.23, “john”, 70.2)
```

https://www.geeksforgeeks.org/python-difference-between-list-and-tuple/

Tuples

```
>>> print(tinytuple * 2)          # Prints tuple two times
(123, “john”, 123, “john”)
>>> print(tuple + tinytuple)      # Prints concatenated tuples
(“abcd”, 786 , 2.23, “john”, 70.2, 123, “john”)
```

https://www.geeksforgeeks.org/python-difference-between-list-and-tuple/

Combination

List of tuples

list = [(“oranges”, 10), (“apples”, 20)]

```
>>> listTuple = [("oranges", 10), ("apples", 20)]                    # List of tuples
>>> print(listTuple)
[('oranges', 10), ('apples', 20)]
>>> listList = [["oranges", 10], ["apples", 20]]                     # List of lists
>>> print(listList)
[['oranges', 10], ['apples', 20]]
>>> tupleList = (["oranges", 10], ["apples", 20])                    # Tuple of lists
>>> print(tupleList)
(['oranges', 10], ['apples', 20])
>>> tupleTuple = (("oranges", 10), ("apples", 20))                   # Tuple of tuples
>>> print(tupleTuple)
(('oranges', 10), ('apples', 20))
```

List of list,
tuple of lists,
tuple of tuples

Dictionaries

```
>>> dictionary = {}
>>> dictionary ["one"] = "This is one"
>>> dictionary [2]       =  "This is two"
>>> tinydictionary      = { "name": "john", "code": 6734, "dept": "sales"}
>>> print(dictionary ["one"])              # Prints value for "one" key
This is one
>>> print(dictionary [2])           # Prints value for 2 key
This is two
```

Dictionaries

```
>>> print(tinydictionary)          # Prints complete dictionary
{“dept”: “sales”, “code”: 6734, “name”: “john”}
>>> print(tinydictionary.keys())          # Prints all the keys
[“dept”, “code”, “name”]
>>> print(tinydictionary.values())                # Prints all the values
[“sales”, 6734, “john”]
```

Decision making

```
if expression:
__statement(s)
elif expression:
__statement(s)
else:
__statement(s)
```

```
if value == 0:
    print("The value is 0")
elif value > 0:
    print("The value is greater than 0")
else:
    print("The value is less than 0")
```

Loops

- A loop statement allows us to execute a statement or group of statements multiple times

- Python has two types of loops:
  1. While
  2. For

While

```
while expression:
__statement(s)
```

```
count = 0
while count < 7:
    print("The count is:", count)
    count = count + 1
print("Goodbye!")
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
Goodbye!
```

For

```
for variable in sequence:
__statement(s)
```

```
lang = [“Python”, “Java”, “Ruby”]
for x in lang:
    print(x)
for x in range(0, 5):
    print(“Sequence:”, x)
```

```
Python
Java
Ruby
Sequence: 0
Sequence: 1
Sequence: 2
Sequence: 3
Sequence: 4
```

For

```
logins = [(“DACS”, “dacs_pass”), (“FSE”, “fse_pass”)]

for (login, password) in logins:
    print(login, password)
```

```
DACS dacs_pass
FSE fse_pass
```

Functions

```
def functionname(parameters):
__function_suite
__return expression

def printme(str):
   return “Print this: ” + str

var = printme(“Awesome Python!”)
print(var)
```

Print this: Awesome Python!

Functions

Class

```
class Customer():
    balance = 0;
    name = “default”

    def __init__(self, name, balance=0):
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        return self.balance
```

Attributes

Constructor

```
person = Customer(“Enrique”, 100)
amount = person.deposit(50)
```

Method

Example

- Fibonacci sequence

```
def fib (number):
    if (number<2):
        return number
    else:
        return fib(number-2) +fib(number-1)

for i in range( 20):
    print(fib(i))
```

Comparisons

Python vs Java: comparisons

Example 3: Dynamically typed

Java

```
public class one
{
    public static void main (String[] args)
    {
            int x = 8;
            x = "eight";
    }
}
```

```
Main.java:6: error: incompatible types:
String cannot be converted to int
            x = "Eight";
                ^
1 error
compiler exit status 1
```

Python

```
x = 8                   # x is an integer
print(type(x))
x = "eight"             # x is a string
print(type(x))
```

```
<type 'int'>
<type 'str'>
```

Python vs Java: comparisons

Example 4: Verbosity/Simplicity

Java

```
public class Main
{
    public static  void main (String[] args)
    {
            int x=10, y=20, tmp;
            tmp=x;
            x=y;
            y=tmp;
            System.out.println(x + " " + y);
    }
}
```

Python

```
a, b = 2, 3
a, b = b, a
print(a, b)
```

Python vs Java: comparisons

Example 5: Conditional statements

Java

```
public class HelloWorld
{
    public static  void main(String[] args)
    {
            if (args[0] > 0)
                System .out.println("Positive");
            else if (args[0] < 0)
                System .out.println("Negative");
            else
                System .out.println("Zero");
    }
}
```

Python

```
if (sys.argv[1]>0):
            print("Positive")
elif (sys.argv[1]<0):
            print("Negative")
else
            print("Zero")
```

Python vs Java: comparisons

Example 6: Concatenation of Strings

Java

```
System.out.println("The value is: " + myVariable);
```

Python

```
print("The value is: " + str(myVariable))
```

Python

```
print("The value is:", str(myVariable))
print("The value is: %d" %myVariable)
print("The value is: {}".format(myVariable))
```

Python vs Java: comparisons

Example 7: Looping arrays/lists

Java

```
public class one
{
    public static  void main (String[] args)
    {
            int[] numbers = {1, 2, 3};

            for (int i=0; i<numbers.length; i++)
           System    .out.println(numbers[i]);
    }
}
```

Python

```
numbers = [1, 2, 3]
for numbers_value in numbers:
        print(numbers_value)
```

Python vs Java: comparisons

Example 9: Inheritance

Java

```
public class BankAccount
{
        […]
}

public class SeniorAccount extends BankAccount
{
        […]
}
```

Python

```
class BankAccount:
        […]

class SeniorAccount(BankAccount):
        […]
```

Learning goals

- Identify the main similarities and differences between *Java* and *Python* (e.g. compiler vs interpreter, statically vs dynamically typed, etc.)
- Be able to implement solutions for *basic problems* using Python
- Understand and be able to utilize *variable types* not available in *Java* (or not covered in CS1-CS2)