Python Review

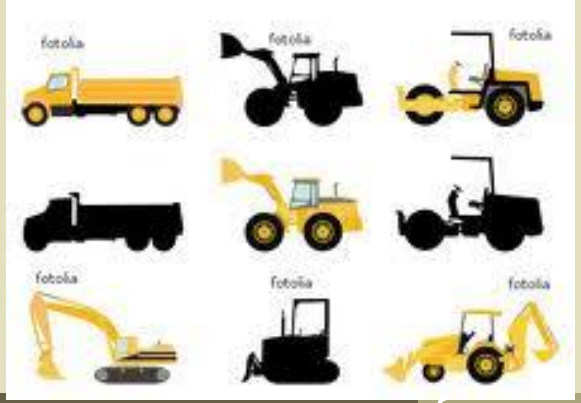
Il-Chul Moon
Dept. of Industrial and Systems Engineering
KAIST
icmoon@kaist.ac.kr

Weekly Objectives

- This week, we review Python that will be used for a programming language for this course.
- Objectives are
  - Setting up the Python development environment
  - Executing “Hello world” program
  - Memorizing basic grammar of Python
  - Understanding Python programming structure
  - Understanding the reference of Python variables
  - Understanding control and loop statements
  - Understanding function calls and class declarations
  - Understanding list, tuple, and dictionary in Python
  - Executing a sample GUI program

Programming and DS&A

- What is programming to data structure and algorithm?
  - Programming is an implementation tool
  - Conceptual thinking and design
    - Where to put the restroom
    - How to find the restroom
  - Practical design and implementation
    - What to use for designing the restroom
    - How to move to the restroom
- Both are important
  - Should pursue good design and good implementation
  - Good design and bad implementation?
  - Bad design and good implementation?



Python

- Python
  - Introduced by Guido van Rossum in 1991
  - Interpreter
  - Object-oriented
  - Dynamic type of variables
  - Increasing usage by industry and academia
  - Unique code structure
    - Mandatory indentation…
  - Fast development speed, slow execution speed
  - Specialty in data analyses
    - Various numerical and statistical libraries : NumPy and SciPy
    - Base language for TensorFlow and others

Programming and Execution Environment

- You will need an integrative development environment (IDE)
  - To reduce implementation time
  - To reduce debugging time
  - To maintain consistencies between your classmates
- Software
  - Eclipse – IDE software
    - PyDev – Eclipse plugin for Python programming
  - Python – Programming Language
  - PyCharm – IDE Software
- Refer to the below website for the setup
  - http://pydev.org/manual_101_root.html

Hello World in Python

- Your first Python program in this class
- Procedure-oriented program
  - main() is a function
- Largely in two parts,
  - Definition part
  - Execution part

```python
def main():
    print("Hello, world!")
    print("This program computes the average of two exam scores.")

    score1, score2 = input("Enter two scores separated by a comma: ").split(",")
    average = (int(score1) + int(score2)) / 2.0

    print("The average of the scores is : ", average)


main()
```

```
HelloWorld
C:₩Users₩USER₩coding_new₩Scripts₩python.exe C:/Users/USER/Desktop/IE260/coding_ne
Hello, world!
This program computes the average of two exam scores.
Enter two scores separated by a comma: 1,3
The average of the scores is :  2.0
```

Python Program Structure - Another Hello World

- Your second Python program in this class
- Object-oriented program
  - HelloWorld is an object
  - __init__, __del__, and performAverage are methods
- Largely in two parts
  - Definition part
  - Execution part

```python
class HelloWorld:
    def __init__(self):  # constructor
        print("Hello World! Just one more time")

    def __del__(self):
        print("Good bye!")

    def performAverage(self, val1, val2):
        average = (val1 + val2) / 2.0
        print("The average of the scores is : ", average)


def main():
    world = HelloWorld()
    score1, score2 = input("Enter two scores separated by a comma: ").split(",")
    world.performAverage(int(score1), int(score2))


main()
```

AnotherHelloWorld

```
C:\Users\USER\coding_new\Scripts\python.exe C:/Users/USER/Desktop/IE260/coding_ne
Hello World! Just one more time
Enter two scores separated by a comma: 1,3
The average of the scores is :  2.0
Good bye!
```

Naming and Styling

```python
class HelloWorld:
    def __init__(self): # constructor
        print("Hello World! Just one more time")

    def __del__(self):
        print("Good bye!")

    def performAverage(self, val1, val2):
        average = (val1 + val2) / 2.0
        print("The average of the scores is : ", average)


def main():
    world = HelloWorld()
    score1, score2 = input("Enter two scores separated by a comma: ").split(",")
    world.performAverage(int(score1), int(score2))
```

- Naming : Use names clearly conveying the meaning
  - Use camel casing
    - Class name : Noun for the concept to be represented by the class
      - Capitalize the first letter of each word
      - e.g. class MyFirstClass:
    - Variable name : Noun for the contents to be stored
      - Start with lower case
      - e.g. numberOfStudents = 100
      - Acceptable, but not recommended in Python
        - e.g. intCount = 0;
    - Method name : Verb for the method action
      - Start with lower case
      - e.g. def performAverage (self,val1,val2):
- Indentation
  - 4 spaces per each level

Comments

is the beginning of single-line comments

```
'''
Created on 2011. 8. 23.
Modified on 2019. 3. 11.

@author: Il-Chul Moon
'''
```

Three of ' is the beginning and the ending of multiline comments

```
"""
Created on 2011. 8. 23.
Modified on 2019. 3. 11.

@author: Il-Chul Moon
"""
```

Three of " is the beginning and the ending of multiline comments

```
"
Created on 2011. 8. 23.
Modified on 2019. 3. 11.

@author: Il-Chul Moon
"""
```

Error!

```
class HelloWorld:
    def __init__(self): # constructor
        print("Hello World! Just one more time")

    def __del__(self):
        print("Good bye!")
```

```
'''
Created on 2011. 8. 23.
Modified on 2019. 3. 11.

@author: Il-Chul Moon
'''


# class declaration begins here
class HelloWorld:
    def __init__(self): # constructor
        print("Hello World! Just one more time")
```

- Writing comments is critical
  - For TA
    → your future boss
  - For your friends
    → your future colleagues
  - For yourself
    → yourself in future
- Different types of comments

Data Types

Classification	Data type	Description and examples
Numeric Data Types	integer	signed integer, 32 bits, 2147483647
	float	64 bit double precision, like 1.23 or 7.8e-28
	long integer	arbitrarily large integer, trailing L, 234187626348292917L, 7L
	octal integer	base-8 integer, leading 0 as in 0177
	hexadecimal integer	base-16 integer, leading 0x as in 0x9FC
	complex	real and imaginary parts written as 3 + 4j or 1.23 - 0.0073j
String Data Types	character string	ordered collection of characters, enclosed by pairs of ', or " characte rs
Collection Data Types	list	ordered collection of objects, like [1,22,[321,'cow'],'horse']
	dictionary	collection of associated key:data pairs like {'first':'alpha','last':'omega'}
	tuples	similar to lists, like ('hen','duck',('rabbit','hare'),'dog','cat')

Variable Statements

Integer in decimal, octal, and hexadecimal

Complex numbers

See the naming style

```python
def main():
    numYearBase10 = 2019
    numYearBase8  = 0o3743
    numYearBase16 = 0x7E3

    print ("Year by base 10 : %d, by base 8 : %d, by base 16 : %d" % (numYearBase10, numYearBase8, numYearBase16))

    numComplex1 = complex(3, 4)
    numComplex2 = 4+3j

    print ("Complex value : ", numComplex1)
    print ("Absolute value : ", abs(numComplex2))
    print ("Real value : ", numComplex2.real)
    print ("Image value : ", numComplex2.imag)

    strDeptName = "Industrial & Systems Engineering"
    strUnivName = "KAIST"
    print ("Department : ", strDeptName)
    print ("Full name of dept. : ", (strDeptName+", "+strUnivName))

main()
```

3_VariableStatements

```
C:₩Users₩USER₩coding_new₩Scripts₩python.exe C:/Users/USER/Desktop/IE260/coding_new/src/edu/kaist/seslab/ie362/week1
Year by base 10 : 2019, by base 8 : 2019, by base 16 : 2019
Complex value :  (3+4j)
Absolute value :  5.0
Real value :  4.0
Image value :  3.0
Department :  Industrial & Systems Engineering
Full name of dept. :  Industrial & Systems Engineering, KAIST
```

Operators

```
def main():
    numTest1 = 10
    numTest2 = 3.0
    numPlus = numTest1 + numTest2
    numMinus = numTest1 - numTest2
    numMultiply = numTest1 * numTest2
    numDivide = numTest1 / numTest2
    numModula = numTest1 % numTest2
    print("%d, %d, %d, %f, %d" % (numPlus, numMinus, numMultiply, numDivide, numModula))

    numDivideInt = numTest1 / int(numTest2)
    print(numDivide, numDivideInt)

    numTest2, numTest1 = numTest1, numTest2
    print(numTest1, numTest2)

    print(numTest1 == numTest2)
    print(numTest1 != numTest2)
    print(type(numTest1))

    numTest1 = str(numTest1)
    print(type(numTest1), numTest1)

    strFormula = "2011 / 7"
    print(eval(strFormula))
main()
```

```
13, 7, 30, 3.333333, 1
3.3333333333333335 3.3333333333333335
3.0 10
False
True
<class 'float'>
<class 'str'> 3.0
287.2857142857143
```

+, -, *, /
Then %?
It is the modulo operator

See the effect of type casting and how-to int(*variable name*)

See the swapping statement

== is the equivalence of values
!= is the in-equivalence of values

String Formula evaluation

String

H	e	l	l	o		W	o	r	l	d	!		I	S	E
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
strTest = 'Hello World! ISE'
print(strTest)
strTestComp = "Hello World! ISE"
print(strTestComp, strTest == strTestComp)

print(strTest[0], strTest[1])
print(strTest[-1], strTest[-2])

print(len(strTest))
print(strTest+" "+"Dept")
print(strTest*2)
print("ISE" in strTest)
print("ISE" not in strTest)
```

```
Hello World! ISE
Hello World! ISE True
H e
E S
16
Hello World! ISE Dept
Hello World! ISEHello World! ISE
True
False
```

String variable statements
Both of ‘ and “ work as wrappers

String value equivalence test, quite simple!

String variable is actually a linear collection of letters, and the letters have indexes

See how the string operators work!

Existence check in collection variables

Index in Sequence

This index applies to strings
as well as tuples, lists
→ Applies to any sequence variables

Simple index of a sequence, or an array

```
strTest = 'Hello World! ISE'
print(strTest)
print(strTest[1], strTest[2], strTest[3])
print(strTest[1:3])
print(strTest[3:])
print(strTest[:3])
print(strTest[1:9:2])
print(strTest[1:len(strTest):2])
print(strTest[1::2])
print(strTest[5::-1])
```

```
Hello World! ISE
e l l
el
lo World! ISE
Hel
el o
el ol!IE
el ol!IE
olleH
```

x:y → from x to y

x:y:z → from x to y with z steps

Default:
y = the length of the sequence
z = 1

List

List is another type of sequence variables

```
lstTest = [1, 2, 3, 4]
print(lstTest)
print(lstTest[0], lstTest[1], lstTest[2])
print(lstTest[-1], lstTest[-2])
print(lstTest[1:3])
print(lstTest+lstTest)
print(lstTest+3)

lstTest = list(range(1, 20, 3))
print(lstTest)
print(4 in lstTest, 100 in lstTest)
lstTest.append('hey')
print(lstTest)
del lstTest[0]
print(lstTest)
lstTest.reverse()
print(lstTest)
lstTest.remove(4)
print(lstTest)
```

```
[1, 2, 3, 4]
1 2 3
4 3
[2, 3]
[1, 2, 3, 4, 1, 2, 3, 4]
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
[1, 4, 7, 10, 13, 16, 19]
True False
[1, 4, 7, 10, 13, 16, 19, 'hey']
[4, 7, 10, 13, 16, 19, 'hey']
['hey', 19, 16, 13, 10, 7, 4]
['hey', 19, 16, 13, 10, 7]
```

See how the operators work

range(x,y,z) == x:y:z
You will use this function many, many times

in and *not in* comes pretty handy

Tuple

- Tuple and List are almost alike
- Only different in changing values
  - Tuple does not allow value changes

```
tplTest = (1, 2, 3)
print(tplTest)
print(tplTest[0], tplTest[1], tplTest[2])
print(tplTest[-1], tplTest[-2])
print(tplTest[1:3])
print(tplTest+tplTest)
print(tplTest*3)

tplTest[0] = 100


(1, 2, 3)
1 2 3
3 2
(2, 3)
(1, 2, 3, 1, 2, 3)
(1, 2, 3, 1, 2, 3, 1, 2, 3)
Traceback (most recent call last):
  File "C:/Users/USER/Desktop/IE260/coding_new/src/edu/kaist/seslab/ie362/week1/7_TupleTest.py", line 19, in <module>
    tplTest[0] = 100
TypeError: 'tuple' object does not support item assignment
```

Dictionary

- Dictionary is also a collection variable type
  - However, it is not sequential
  - It works by a pair of keys and values
    - A set of (key 1, value 1), (key 2, value 2), (key 3, value 3)…
    - Exact syntax is { key1:value1, key2:value2, key3:value3 …)

```
dicTest = {1: 'one', 2: 'two', 3: 'three'}
print(dicTest[1])
dicTest[4] = 'four'
print(dicTest)
dicTest[1] = 'hana'
print(dicTest)
print(dicTest.keys())
print(dicTest.values())
print(dicTest.items())
```

```
one
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
{1: 'hana', 2: 'two', 3: 'three', 4: 'four'}
dict_keys([1, 2, 3, 4])
dict_values(['hana', 'two', 'three', 'four'])
dict_items([(1, 'hana'), (2, 'two'), (3, 'three'), (4, 'four')])
```

if

- A condition statement
  - **if** *boolean*:
    Statements for True
    elif *boolean*:
    Statements for True
    else:
    Statements for False
- Python does not have a switch-case statement
  - You will have to live with *ifs*
- Watch your indentations carefully because that is your block statements

```
numScore = 95
if numScore > 90:
    print('A')

numScore = 75
if numScore > 90:
    print('A')
else:
    print('Lower grade')

if numScore > 90:
    print('A')
elif numScore > 80:
    print('B')
elif numScore > 70:
    print('C')
else:
    print('D or F')
```

```
A
Lower grade
C
```

for

- A loop statement
- The most common loop statement in programming languages
  - **for** *variable* in *sequence*:

    Statements for loop

    else:

    when for-loop is finished without a break
- Some useful statements for loops
  - *continue*
  - *break*

```
for itr in range(10):
    print(itr, end=' ')
print()

sum = 0
for itr in range(1, 11):
    sum += itr
print(sum)

for itr in range(1, 100, 10):
    if itr == 51:
        continue
    else:
        print(itr, end=' ')
print()

for itr in range(5):
    print(itr, end=' ')
else:
    print('!')
print('done')

for itr in range(5):
    if itr == 3:
        break
    print(itr, end=' ')
else:
    print('!')
print('done')
```

```
0 1 2 3 4 5 6 7 8 9
55
1 11 21 31 41 61 71 81 91
0 1 2 3 4 !
done
0 1 2 done
```

while

- Second loop statement
- Syntax is
  - **while** *boolean*:

    Statements for loop

    else:

    when for-loop is finished without a break
- Still you can use the two loop control statements
  - *continue*
  - *break*

```
itr = 0
sum = 0
while itr < 10:
    itr = itr + 1
    sum = sum + itr
else:
    print("Sum from 1 to 10 : ", sum)

itr = sum = 0
while itr < 10:
    itr = itr + 1
    if itr == 5:
        break
    sum = sum + itr
else:
    print("Sum from 1 to 10 : ", sum)
print("Good bye")
```

```
Sum from 1 to 10 :  55
Good bye
```

Function Statement

- **def** *name*(*params*)**:**

  statements

  return *var1, var2…*
- You can return multiple variables
  - Keep them in order
- You do not have to specify return types
  - Anyway you don't have types in Python
- One line function is called *lambda* function

```
numA = 1
numB = 2

def add(numParam1, numParam2):
    return numParam1 + numParam2

def multiply(numParam1, numParam2):
    return numParam1*2, numParam1*3

def increase(numParam1, step = 1):
    return numParam1+step

numC = add(numA, numB)
numD, numE = multiply(numA, numB)
numF = increase(numA, 5)
numG = increase(numA)

lambdaAdd = lambda numParam1, numParam2 : numParam1 + numParam2

numH = lambdaAdd(numA,numB)

print(numC, numD, numE, numF, numG, numH)
```

```
3 2 3 6 2 3
```

Sample Program: Finding Prime Numbers

```
def isPrimeNumber(numParam1):
    for itr in range(2, numParam1):
        if numParam1 % itr == 0:
            break
    else:
        return True
    return False


def findPrimes(numParam1, numParam2):
    numCount = 1
    for itr in range(numParam1, numParam2):
        if isPrimeNumber(itr) == True:
            print(numCount, " th prime : ", itr)
            numCount = numCount + 1


findPrimes(1, 10)
```

Function for calculation

Function for iteration

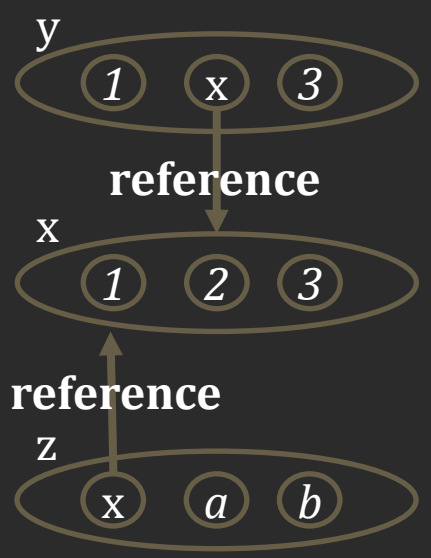
Triggers the execution

Statement for execution

```
1  th prime :  1
2  th prime :  2
3  th prime :  3
4  th prime :  5
5  th prime :  7
```

Assignment and Equivalence

```python
x = [1, 2, 3]
y = [100, x, 120]
z = [x, 'a', 'b']

print('x : ', x)
print('y : ', y)
print('z : ', z)

x[1] = 1717

print('₩nx : ', x)
print('y : ', y)
print('z : ', z)

x[1] = 2
x2 = [1, 2, 3]
if x == x2:
    print("Values are equivalent")
else:
    print("Values are not equivalent")

if x is x2:
    print("Values are stored at the same place")
else:
    print("Values are not stored at the same place")

if x[1] is y[1][1]:
    print("Values are stored at the same place")
else:
    print("Values are not stored at the same place")
```




```
x :  [1, 2, 3]
y :  [100, [1, 2, 3], 120]
z :  [[1, 2, 3], 'a', 'b']

x :  [1, 1717, 3]
y :  [100, [1, 1717, 3], 120]
z :  [[1, 1717, 3], 'a', 'b']
Values are equivalent
Values are not stored at the same place
Values are stored at the same place
```

- One variable's value is changed
- But, you see three changes
- Why this happened?
  - Because of references
  - x has two references from y and z
  - The values of y and z are determined by x, and x is changed
    - See the ripple effects
- ***==***
  - Checks the equivalence of two referenced values
- ***is***
  - Checks the equivalence of two referenced objects' IDs

References, Symbol Table, and Object Table

```python
import sys

x = [1, 2, 3]
y = [100, x, 120]
z = [x, 'a', 'b']

print(sys.getrefcount(x))
print(sys.getrefcount(y))

print(id(x))
print(id(y[1]))
print(id(y))

print(x is y[1])
print(x is y)
```

```
4
2
2490265005512
2490265005512
2490265003464
True
False
```

ID(*variable*) returns the referenced object ID
ID(*variable1*)==ID(*variable2*)
→ *variabl1* **is** *variable2*

sys.getrefcount(*variable*) returns the number of references of an object ID + 1

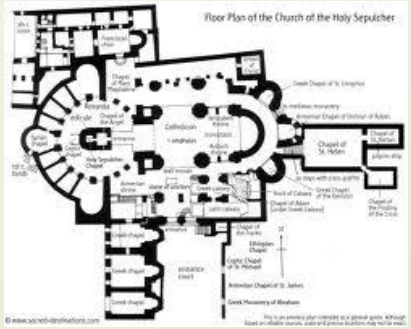
Symbol Table

Var.	x	y	z

Object Table

ID	3887 2808	…	…	…	…	…	…	…	3887 3048	…
Value	List object named x	1	2	3	100	120	‘a’	‘b’	List object named y	List object named z

Class and Instance



instantiation

```
class MyHome:
    colorRoof = 'red'
    stateDoor = 'closed'
    def paintRoof(self,color):
        self.colorRoof = color
    def openDoor(self):
        self.stateDoor = 'open'
    def closeDoor(self):
        self.stateDoor = 'close'
    def printStatus(self):
        print ("Roof color is", self.colorRoof, ", and door is", self.stateDoor)


homeAtDaejeon = MyHome()
homeAtSeoul = MyHome()
homeAtSeoul.openDoor()
homeAtDaejeon.paintRoof('blue')
homeAtDaejeon.printStatus()
homeAtSeoul.printStatus()
```

See how to define a class → **class** *classname*:

See how to instantiate a class → *var* = *classname*(*param*)

```
Roof color is blue , and door is closed
Roof color is red , and door is open
```

Important Methods in Class - Constructor, Destructor

- Some basic methods, or member functions in classes
  - Constructor
    - Called when instantiated
  - Deconstructor
    - Called when the instance is removed from the value table

```python
from time import ctime

class MyHome:
    colorRoof = 'red'
    stateDoor = 'closed'
    def paintRoof(self,color):
        self.colorRoof = color
    def openDoor(self):
        self.stateDoor = 'open'
    def closeDoor(self):
        self.stateDoor = 'close'
    def printStatus(self):
        print("Roof color is", self.colorRoof, ", and door is", self.stateDoor)
    def __init__(self, strAddress):
        print("Built on", strAddress)
        print("Built at", ctime())
    def __del__(self):
        print("Destroyed at", ctime())


homeAtDaejeon = MyHome('Daejeon KAIST')
homeAtDaejeon.printStatus()
del homeAtDaejeon
```

```
Built on Daejeon KAIST
Built at Mon Mar 11 14:04:33 2019
Roof color is red , and door is closed
Destroyed at Mon Mar 11 14:04:33 2019
```

Module and Import

Home.py

```python
'''
Created on 2011. 8. 24.
Modified on 2019. 3. 11.

@author: Il-Chul Moon
'''

from time import ctime

class MyHome:
    colorRoof = 'red'
    stateDoor = 'closed'
    def paintRoof(self,color):
        self.colorRoof = color
    def openDoor(self):
        self.stateDoor = 'open'
    def closeDoor(self):
        self.stateDoor = 'close'
    def printStatus(self):
        print ("Roof color is", self.colorRoof, ", and door is", self.stateDoor)
    def __init__(self, strAddress):
        print("Built on", strAddress)
        print("Built at", ctime())
    def __del__(self):
        print("Destroyed at", ctime())
```

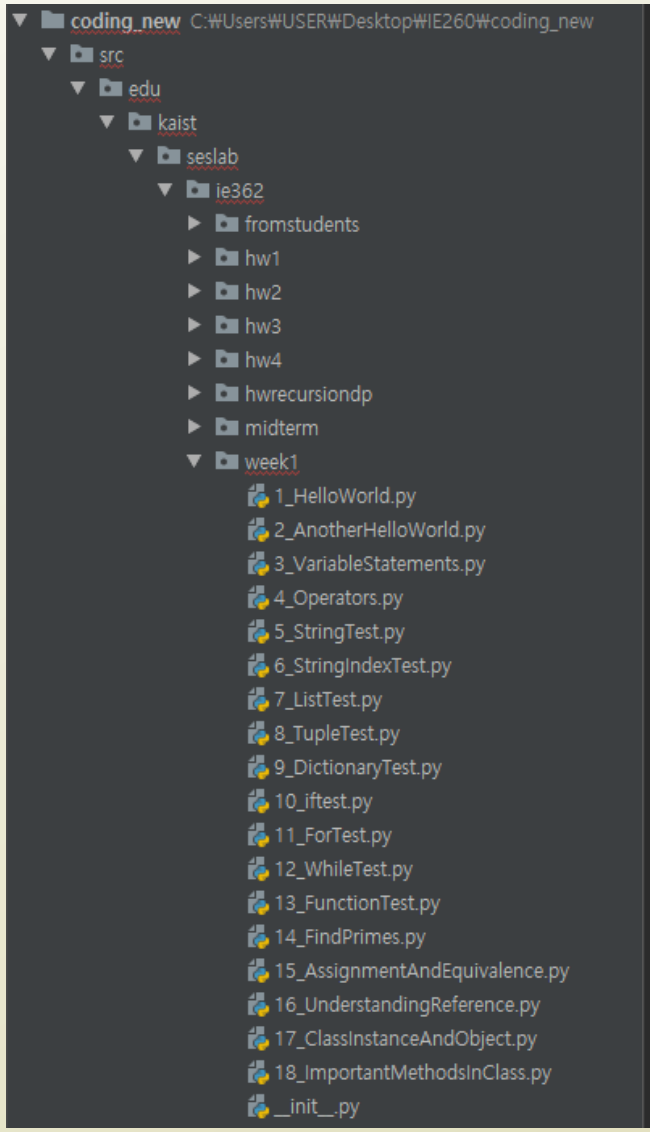
Home.py / UsingMyHome.py

```python
'''
Created on 2011. 8. 24.
Modified on 2019. 3. 11.

@author: Il-Chul Moon
'''

from src.edu.kaist.seslab.ie362.week1 import Home

homeAtDaejeon = Home.MyHome("KAIST Daejeon")
homeAtDaejeon.printStatus()
```

```
Built on KAIST Daejeon
Built at Mon Mar 11 14:10:52 2019
Roof color is red , and door is closed
Destroyed at Mon Mar 11 14:10:52 2019
```

- See how to separate the source code files
  - Just put your code in another file
  - *filename*.**py**
- See how to use classes in other files
  - **import** *filename*
- Use from to specify the directory, or the folder, path

Organizing Modules by Package




```
from src.edu.kaist.seslab.ie362.week1 import Home

homeAtDaejeon = Home.MyHome("KAIST Daejeon")
homeAtDaejeon.printStatus()
```

- Directory, or folder
  - Clusters modules
    - Modules → filename.py
  - We call these directories as package
  - Hence, the previous information is exactly
    - from *package* **import** *module*
- Package has
  - __init__.py in the directory
  - This is how to differentiate between the ordinary and the package directories

Sample Program: Interaction with Your Program

```python
class CashierLine:
    lstLine = []
    def addCustomer(self, strName):
        self.lstLine.append(strName)
    def processCustomer(self):
        strReturnName = self.lstLine[0]
        self.lstLine.remove(strReturnName)
        return strReturnName
    def printStatus(self):
        strReturn = ""
        for itr in range(len(self.lstLine)):
            strReturn += self.lstLine[itr] + " "
        return strReturn


binLoop = True
line = CashierLine()
while binLoop:
    strName = input("Enter customer name :")
    if strName == ".":
        break
    elif strName == "->":
        print("Processed :", line.processCustomer())
        print("Line :", line.printStatus())
    else:
        line.addCustomer(strName)
        print("Line :", line.printStatus())
print("Number of remaining customers :", len(line.lstLine))
```

```
Enter customer name :Moon
Line : Moon
Enter customer name :Sun
Line : Moon Sun
Enter customer name :->
Processed : Moon
Line : Sun
Enter customer name :.
Number of remaining customers : 1
```

Member variable

See how to define a member function
def *funcname*(**self,** *param*)**:**

Add *self* when accessing member variables

Iterating the member variable to print out who are in the line

Your first interaction with a python program in this course

Sample Program: GUI in Python

There will be chances to cover more on GUI in the future of this course

- GUI in Python
  - Need to import many modules from tkinter package
  - Statements
    - Instantiate and place your GUI items
      - Text
      - Label
      - Button
    - You can link a function to the items at the instantiation timing
      - “command=paintRoof”
    - pack()
      - Make the items to fit the specified size
    - root.mainloop()
      - For you to see the dialog window
      - Make the window hold

```
from tkinter import *
from src.edu.kaist.seslab.ie362.week1 import Home

homeDaejeon = Home.MyHome("Daejeon KAIST")

def paintRoof():
    strColor = txtColor.get(1.0, END)
    homeDaejeon.paintRoof(strColor)
    lblColor['text'] = homeDaejeon.colorRoof


root = Tk()

txtColor = Text(root, width=20, height=2)
lblColor = Label(root, text=homeDaejeon.colorRoof, width=20, height=2)
btnColor = Button(root, text="Paint the Roof", width=20, height=1,
                  command=paintRoof)
txtColor.pack()
lblColor.pack()
btnColor.pack()

root.mainloop()
```