

# DataSetructure & Algorithm

1강

## 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

- **Dynamic typing**
  - 변수를 사용할 때 변수 타입을 따로 지정하지 않고 실행할 때 변수의 타입을 자동으로 검사한다.
- **Script Language**
  - 컴파일을 하지 않고 인터프리터가 코드를 직접 한 줄 씩 실행하는 방식이다.
- **Multi paradigm**
  - 절차적, 객체지향적, 함수형, 관점형 프로그래밍이 모두 가능하다.
- **Unlimited access**
  - 사용의 단순함과 편리함을 위해 접근제어 없이 객체, 구조체 member에 무제한적 접근이 가능하다.
- **Everything is object**
  - 변수와 함수 모두가 거의 다 객체다.

# Python Naming and Styling

## ◆ 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

Naeming = 의미를 잘 전달할 수 있는 단어로

camel case = `myComputer` , `helloWorldPython`

Class의 첫 글자는 대문자

변수에는 어떤 데이터가 저장되어있는지 잘 나타낼 수 있는 명사 단어로 표현하고 첫 글자는 소문자

메소드이름은 동사로 표현하고 첫 글자는 소문자

들여쓰기는 기본적으로 4개의 띄어쓰기

# Index

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

```
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]
```

Simple index of a  
sequence, or an array

x:y → from x to y

x:y:z → from x to y  
with z steps

Default:  
y = the length of the sequence

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

# Python 자료구조 - 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
```

See how the operators  
work

```
lstTest = range(1,20,3)
print lstTest
print 4 in lstTest, 100 in lstTest
lstTest.append('hey')
```

$\text{range}(x,y,z) == x:y:z$

You will use this function many, many  
times

```
print lstTest
del lstTest[0]
print lstTest
lstTest.reverse()
print lstTest
lstTest.remove(4)
```

*in* and *not in* comes  
pretty handy

```
print lstTest
lstTest.sort()
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]
```

[] 대괄호로 정의

$\text{list} + \text{list} = [\text{list}[0] \dots\dots]$  합쳐짐

$\text{list} * 3 = \text{list} + \text{list} + \text{list}$

$\text{range}(x,y,z)$

x부터 y까지 z패턴으로

$\text{range}(1,20,3) = 1,4,7,10,13,16,19$

$\text{range}(y)$

0부터 1씩 증가

$\text{range}(x,y)$

x부터 y까지 1씩 증가

*in* < 으로 속해있는지 boolean값으로 확인

$\text{list.append(val)}$

$\text{del list[index]}$

$\text{list.reverse()}$

$\text{list.remove[val]}$

$\text{list.sort()}$

negative index 가능

-1인덱스는 맨 뒤에 값

# Python 자료구조 - 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
```

()소괄호로 정의

리스트와 차이점은 Value가 변하지않음  
아이템을 수정 할 수 없음

상수를 저장할때 표현

# Python 자료구조 - 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'}  
[1, 2, 3, 4]  
['hana', 'two', 'three', 'four']  
[(1, 'hana'), (2, 'two'), (3, 'three'), (4, 'four')]
```

{ } 중괄호로 정의  
선형적이지 않음

Key Value pair

dict[key] = value 로 값 추가

dict.keys() 키값 리스트로 출력

dict.values() 밸류값 리스트로 출력

dict.items() 키밸류 페어 리스트로 출력



# 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

```
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
```

if

## Condition Statement

내용을 꼭 입력해야하고  
내용이 없으면 **pass** 사용하기

들여쓰기를 잘 확인하기

# 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,
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,
print

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

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

```

```

0 1 2 3 4 5 6 7 8 9
55
1 11 21 31 41 51 61 71 81 91
0 1 2 3 4 !
done

```

## 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
```

**return** 값을 여러개 줄 수 있음

단 받을때 순서에 주의 하면서 받아야함

# Reference

## Assignment and Equivalence

```
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 'x : ', x
print 'y : ', y
print 'z : ', z
```

```
x[1] = 2
x2 = [1, 2, 3]
if x == x2:
    print "Values are equivalent"
```

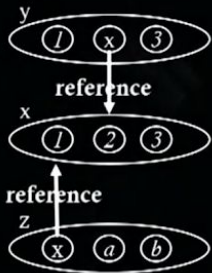
```
else:
    print "Values are not equivalent"
```

```
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

## Reference

다른 언어에서는

call by value와 call by reference 라고 부르는데 파이썬의 함수 인수 전달 방식을 객체 참조에 의한 호출(Call by Object Reference) 또는 공유에 의한 호출(Call by Sharing)이라고 부른다.

==은 변수가 같은 값을 가지고 있으면 true

is는 변수가 같은 객체를 가지고 있으면 true

# Import

## Module and Import

```
Created on 2011. 8. 24.  
@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()
```

```
Created on 2011. 8. 24.  
'''  
@author: Il-Chul Moon  
'''  
import Home  
homeAtDaejeon = Home.MyHome("KAIST Daejeon")  
homeAtDaejeon.printStatus()  
  
Built on KAIST Daejeon  
Built at Wed Aug 24 15:05:57 2011  
Roof color is red , and door is closed  
Destroyed at Wed Aug 24 15:05:57 2011
```

- 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

### Import

다른 파일로 분류되어있는 파이썬 파일중에서 내가 원하는 기능을 가진 파일이 있다면 import 파일명 으로 파일을 import하여 사용하고싶은 함수,메소드를 사용할 수 있습니다

뒤에 as 를 붙이면 제 마음대로 줄여서 사용할 수 있는데 alias(별명)의 약어입니다

필요한 모듈을 import하여 원하는 기능을 사용할 수 있습니다.