Warming Up

What is Programming?

- **Programming** is the process of creating a set of instructions that tell a computer how to perform a task using a programming language

- **Programming** is the process of taking an algorithm and encoding it into a programming language, so that it can be executed by a computer

Real World Problem ➔ Representation ➔ Coding (Programming)

Data Type Data Structure	Algorithm

What is Programming?

x : Width
y : Length
z : Height
l : Loading Zone Volume

```
def volume_sedan(x, y, z):
    return   x * y * z
def volume_truck(x, y, z, l)
    return    x * y * z  +  l


def Test():
   print("My Automobile's volume is:",  volume_sedan(10, 15, 25))
   print("Your PickupTruck's volume is:",  volume_truck(10, 15, 25, 1000))
```

Abstraction

- The Real World Problem: P
- Transform P into AP (Abstract Problem) through Abstraction
- Represent the AP using the given Programming Language
  - Using Basic Data Types, Advanced Data Types, User-defined Data Types
- Solve the AP with Algorithm based on Computational Thinking
  - Defining functions

자동차의 User 사용가능 공간을 알아내라!

User 사용공간 계산에 필요한 데이터:

가로
세로
높이

Representation

x: 가로
y: 세로
z: 높이

Coding

```
def volume_compute(x, y, z):
    return  x * y * z
```

Data Types과 연산

- Basic Data Types
  - Integer
  - Floating Number
  - Boolean
  - Char/String

우리가 익숙한 mathematical notation으로 연산

```
>>> 3 + 4
```

- Collection Data Types
  - List
  - Set
  - Dictionary
  - Tuple

특정 data type에 정의된 function들을 call해서 연산

```
>>> myString =  "S N U"
>>> myString.split()
```

- User-Defined Data Types (Classes)
  - Student
  - Automobile
  - ……

특정 data type에 정의된 function들을 call해서 연산

```
>>> myAuto = Automobile("GM", "2016", "5Door")
>>> myAuto.foo()
```

- Library
  - Math
  - Random
  - ….

특정 library에 정의된 function들을 call해서 연산

```
Ex:   import math
      math..sqrt(4)
```

Basic Data Types vs Collection Data Types

- Int
  - X = 3
- Float
  - X = 3.1
- Char/String
  - X = “b”
  - X = “SNU”
- Boolean
  - X = True

- List
  - X = [3, 7, 2, 1]
- Tuple
  - X = (1, 7, 2, 4)
- Set
  - X = { 3, 6, 1, 2}
- Dictionary
  - X = { “Korea”:”Seoul”, “USA”: “DC”}

Taxonomy of Python Functions

- Python Built-in Functions: input(), print(), len(), abs(), set(), …..

- User Defined Functions:

```
def tri_sum(x, y, z)
    result = x + y +z
    return result
```

- Functions belonging to Python Basic Data Types

```
>>> L = [3, 5, 5]
>>> L.append(4)
```

```
>>> S = {3, 5, 6}
>>> S.remove(5)
```

- Functions belonging to User-Defined Classes

```
>>> class Box(object):
        def  calc_space(self):
            ……
```

```
>>> bbb = Box()
>>> space_value = bbb.calc_space()
```

- Functions from Other Modules (consisting of Functions and Classes)

```
>>>  import sam_module
```

- Functions from Python Standard Library (consisting of Functions and Classes)

```
>>>  import math
```

Python Built-In Functions (68개?)

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Python Built-In Reserved Words (28개?)

and	del	for	is	raise
assert	elif	from	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	yield
def	finally	in		

```
def IsPrime (n):
    if (n < 2):
        return False
    for factor in range(n):
        if (n % factor == 0):
            return False
    return True
```

Python Modules

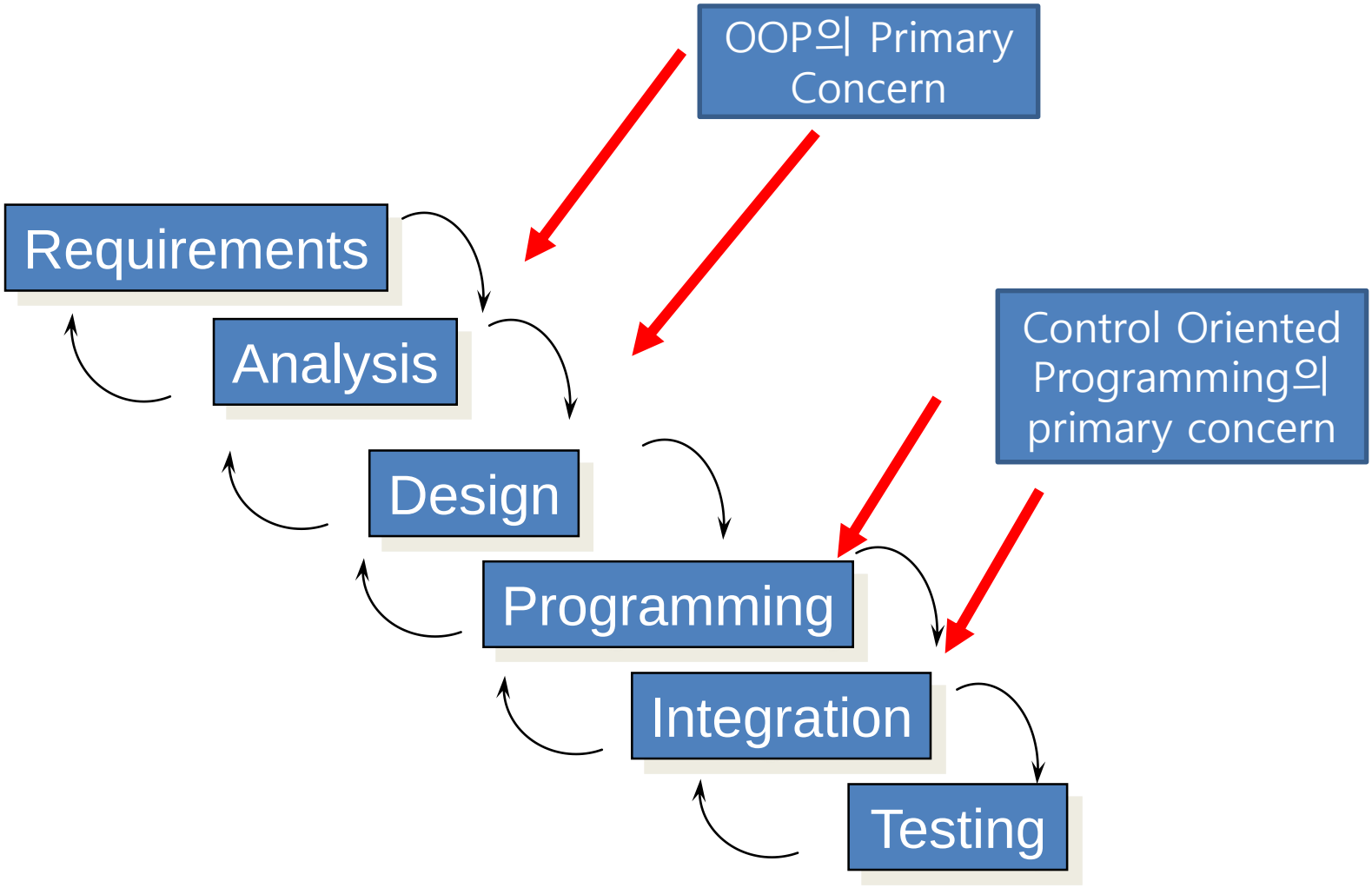
- Standard Library (~ 150여개)
  - Data Types 관련
  - Text Processing 관련
  - File and Directory 관련
  - Arithmetic 관련
  - Basic Statistics 관련
  - Internet 관련
  - Operating System 관련
  - Web Page 관련
  - ….
- Libraries for Various Domains
  - Machine Learning: Numpy, Pandas, MatPlotLib, SK_Learn
  - Scientific Computing: SciPy
  - Game: PyGame
  - Database Connection: MySQLdb
  - …..

High-Level Programming Paradigms

- Control-oriented Programming (before mid 80's)
  - Real world problem ➔ a set of functions
  - Data and functions are separately treated
  - Fortran, Cobol, PL/1, Pascal, C (1972, Bell Lab)

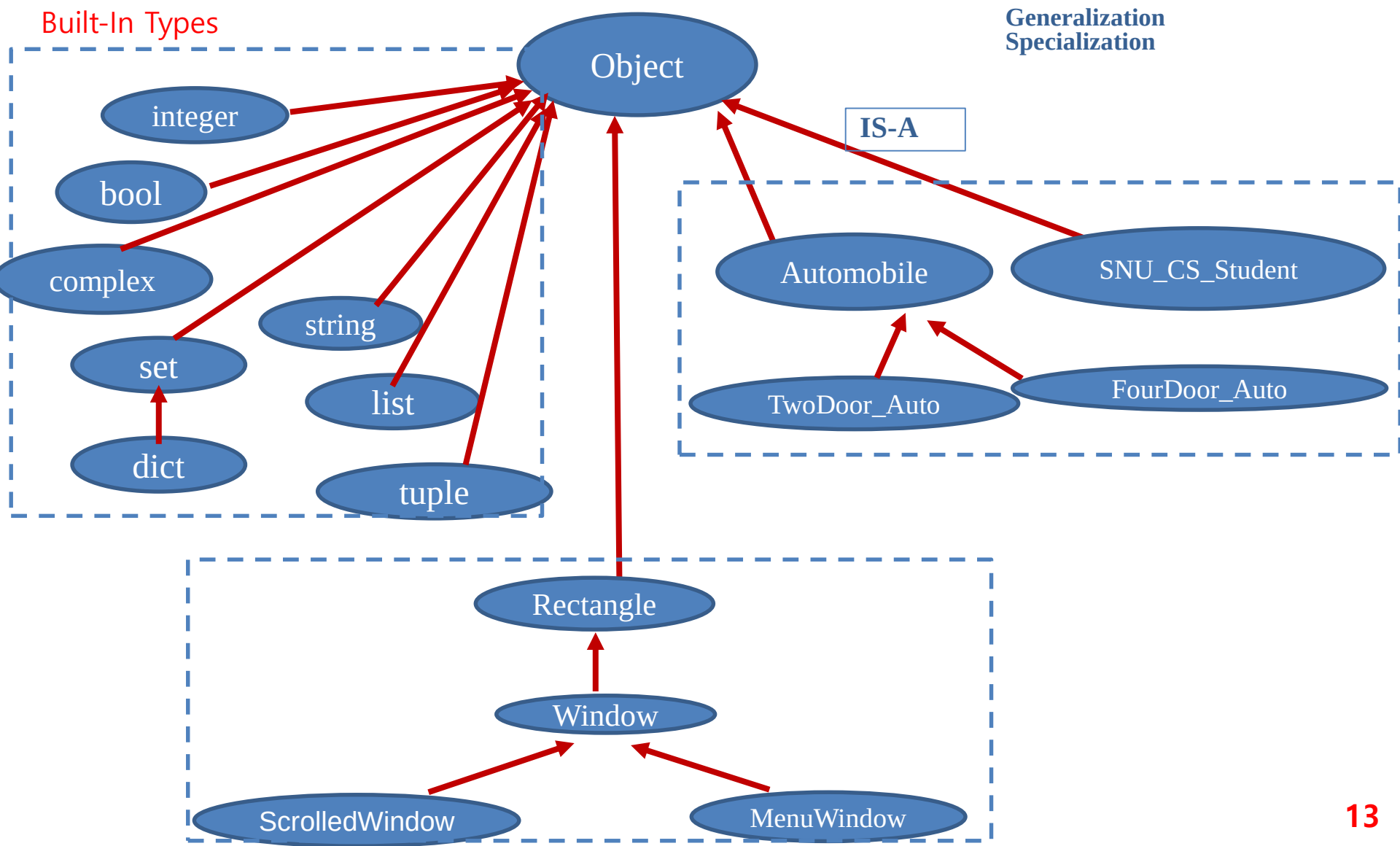
- Object-oriented Programming (after mid 80's)
  - Real world problem ➔ a set of classes
  - Data and functions are encapsulated inside classes
  - C++ (1983, Bell Lab)
  - Python (1991)
  - Java (1993)
  - and most Script Languages (Ruby, PHP, R,…)

Waterfall SW Development Model

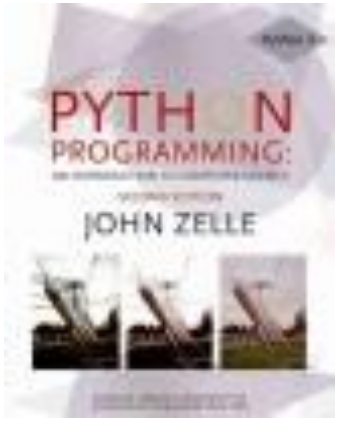


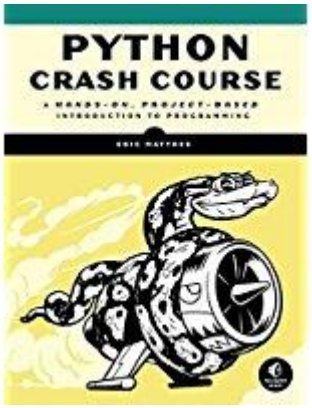
Everything is an Object in OOP

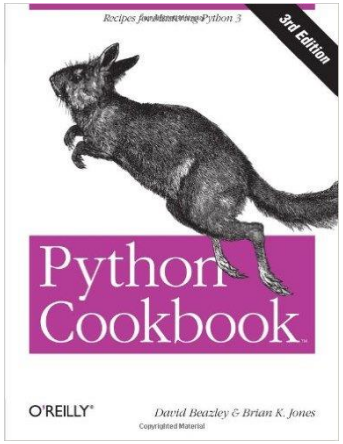
Python의 Type System (class structure)



· Python Books







· Many Many Online Tutorials

https://docs.python.org/3/tutorial/

http://www.python-course.eu/index.php

http://interactivepython.org/courselib/static/thinkcspy/index.html

· Just "Class Notes" + "Googling" is Enough!

https://docs.python.org/3/

Python 3.6.0 documentation

Welcome! This is the documentation for Python 3.6.0, last updated Jan 13, 2017.

Parts of the documentation:

What's new in Python 3.6?
or all "What's new" documents since 2.0

Tutorial
start here

Library Reference
keep this under your pillow

Language Reference
describes syntax and language elements

Python Setup and Usage
how to use Python on different platforms

Python HOWTOs
in-depth documents on specific topics

Installing Python Modules
installing from the Python Package Index & other sources

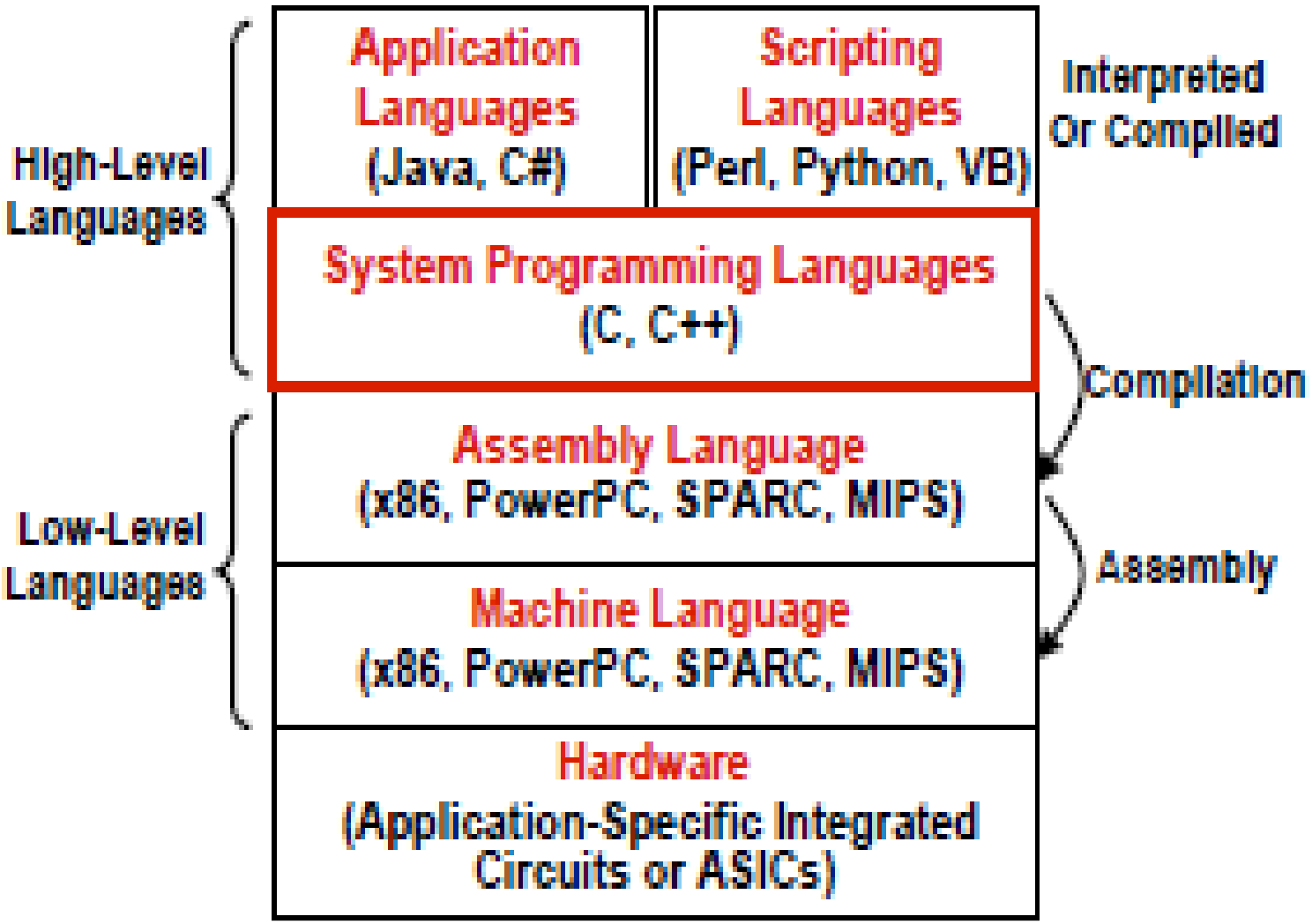
Distributing Python Modules
publishing modules for installation by others

Extending and Embedding
tutorial for C/C++ programmers

Python/C API
reference for C/C++ programmers

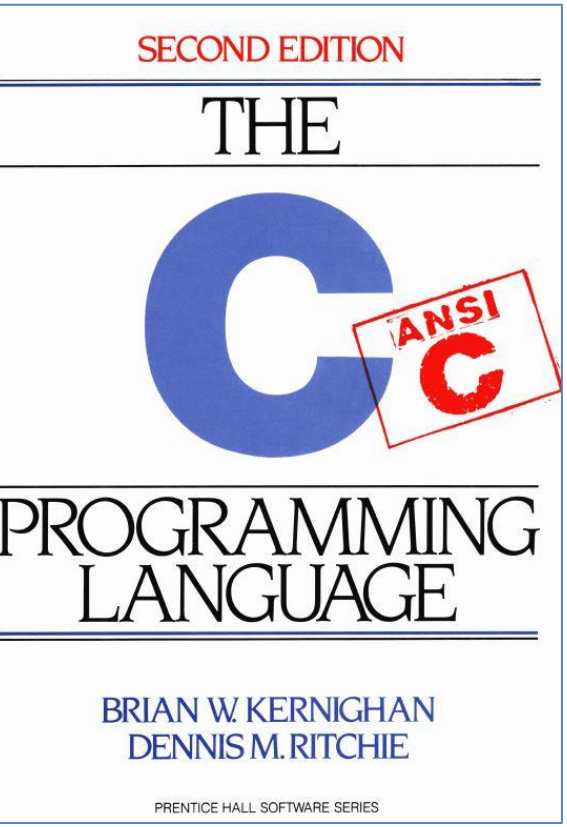
FAQs
frequently asked questions (with answers!)

Programming Levels



1972
by Kerninghan and Richie

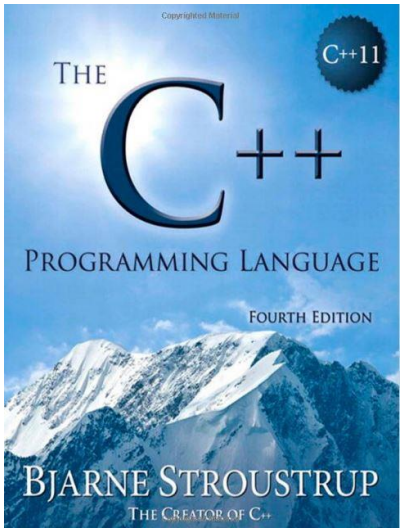
ISO/IEC:
C99 (1999), C11(2011)



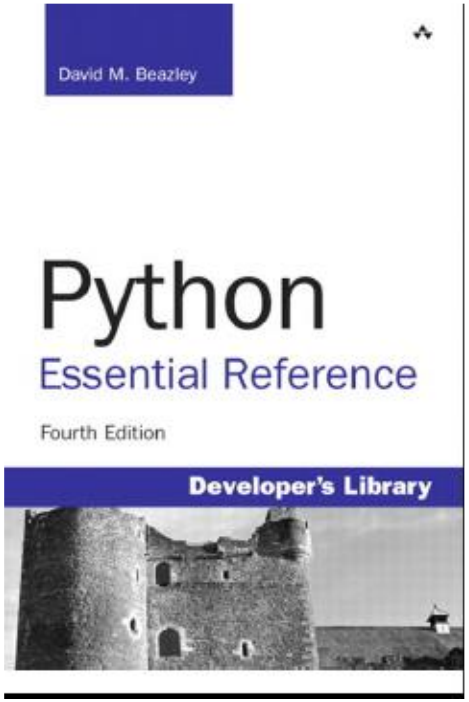
- Chapter 1. Tutorial Introduction
- Chapter 2. Types, Operators, and Expressions
- Chapter 3. Control Flow
- Chapter 4. Functions and Program Structure
- Chapter 5. Pointers and Arrays
- Chapter 6. Structures
- Chapter 7. Input and Output
- Chapter 8. The UNIX System Interface
- Appendix A. Reference Manual
- Appendix B. Standard Library
  - Input and Output: <stdio.h>
  - Character Class Tests: <ctype.h>
  - String Functions: <strings.h>
  - Mathematical Functions: <math.h>
  - Utility Functions: <stdlib.h>
  - Diagnosics: <assert.h>
  - Variable Argument Lists: <stdarg.h>
  - Non-local Jumps: <setjmp.h>
  - Signals: <signal.h>
  - Date and Time Functions: <time.h>
  - Implementation-defined Limits: <limits.h> and <float.h>

C++ 1983 by Bjarne Stroustrup

C++11 (2011)



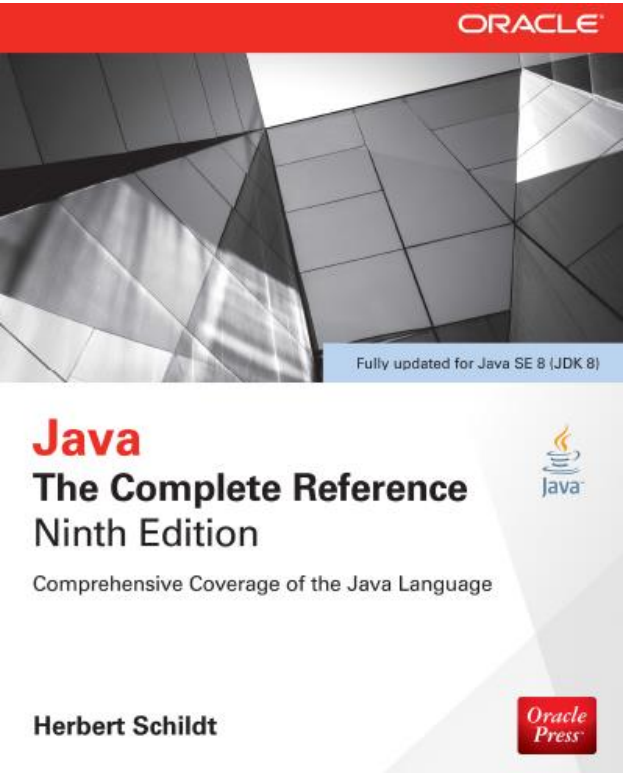
At 1991
by Guido Rossum
Now owned by Python Org
Python 3.6 (2016)



Contents at a Glance

At 1995 by James Gosling
Now owned by ORACLE

Java 8 (2014)



Contents at a Glance

Python I

Computational Thinking with Python

- (7) Python Recursion 47pp
- (8) Python Tuple & Set & Dictionary 45pp
- (9) Python Implementation of List and Set 34pp
- (10) Python Built-In Functions 48pp
- (11) Pyhton File IO and Exceptions 39pp
- (12) Sorting and Searching 50pp
- (13) Math, Stat, Random Modules 24pp
- (14) Simulation 55pp
- (15) OO Paradigm 65pp
- (16) Python OOP 88pp
- (17) Python Module and Package 27pp
- (18) Python Standard Libraries 18pp
- (19) Python DBMS Binding 88pp
- (21) Data Structures 60pp

Python II

BigData Processing with Python

- (22) Stack & Queue Practice 77pp
- (23) AVL & B+ Tree 35pp
- (24) Data Structure 관련 Modules 37pp
- (25) Graphviz Module 70pp
- (26) NetworkX 82pp
- (27) Time, DateTime, Calender Modules 20pp
- (28) System 관련 Modules 53pp
- (29) Python IDEs for Data Analytics 42pp
- (30) Numpy 115pp
- (31) Pandas 112pp
- (32) MatPlotLib Seaborn 92pp
- (34) SK Learn 125pp
- (35) Text Processing Modules 109pp
- (36) NLTK Module 139pp

Review of Python Basic

Python 중급으로 들어가기전 Quick Review!

What is Programming?

- **Programming** is the process of creating a set of instructions that tell a computer how to perform a task using a programming language

- **Programming** is the process of taking an algorithm and encoding it into a programming language, so that it can be executed by a computer

Real World Problem ➔ Representation ➔ Coding (Programming)

Data Type Data Structure	Algorithm

Data Types과 연산

- Basic Data Types
  - Integer
  - Floating Number
  - Boolean
  - Char/String

우리가 익숙한 mathematical notation으로 연산

```
>>> 3 + 4
```

- Advanced Data Types (Collection Data Types)
  - List
  - Set
  - Dictionary
  - Tuple

특정 data type에 정의된 function들을 call해서 연산

```
>>> myString = "S N U"
>>> myString.split()
```

- User-Defined Data Types (Classes)
  - Student
  - Automobile
  - ……

특정 data type에 정의된 function들을 call해서 연산

```
>>> myAuto = Automobile("GM", "2016", "5Door")
>>> myAuto.foo()
```

- Library
  - Math
  - Random
  - ….

특정 library에 정의된 function들을 call해서 연산

```
Ex:   import math
      math..sqrt(4)
```

Basic Data Types

- Integer, Float, Char(String), Boolean

- Types can be probed using "type()" built-in function

```
>>> type(3)
<class 'int'>

>>> type(3.1)
<class 'float'>

>>> type("S")
<class 'string'>

>>> type(True)
<class 'bool'>
```

List Data Type

- Lists are a special kind of *sequence*, so sequence operations also apply to lists!

```
>>> [1,2] + [3,4]
[1, 2, 3, 4]

>>> [1,2]*3
[1, 2, 1, 2, 1, 2]

>>> grades = ['A', 'B', 'C', 'D', 'F']
>>> grades[0]
'A'
>>> grades[2:4]
['C', 'D']
>>> len(grades)
5
```

Program Example with List data type

```
# month.py
#  A program to print the month name, given it's number.
#  This version uses a list as a lookup table.

def main():

    # months is a list used as a lookup table
    months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
              "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

    n = eval(input("Enter a month number (1-12): "))

    print ("The month abbreviation is", months[n-1] + ".")
```

Mutability Issue of List and String Data Type

```
>>> myList = [34, 26, 15, 10]
>>> myList[2]
15
>>> myList[2] = 0
>>> myList
[34, 26, 0, 10]
```

Lists are *mutable* (i.e. they can be changed)

```
>>> myString = "Hello World"
>>> myString[2]
'l'
>>> myString[2] = "p“    #This is not allowed
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    myString[2] = "p"
TypeError: object doesn't support item assignment

>>> myString = "Hi World“     #This is OK
```

Strings are *immutable* : Parts of Strings can **not** be changed using operations

Tuple Data Type

```
>>> t1 = ( )
>>> t2 = (1, )
>>> t3 = (1,  2,  3)
>>> t4 = ('a',  'b',  ['ab',  'cd'])
```

- Tuple is similar to list
  But, Tuple is *immutable*
  Tuple is more memory efficient than List

```
>>> t1 = [ ]
>>> t2 = [1,]
>>> t3 = [1, 2, 3]
>>> t4 = ['a',  'b',  ['ab',  'cd']]
```

```
>>> x = ("ham",4,5)
>>> x
('ham', 4, 5)
>>>

# print(x[2]) is fine
# No add/drop a part inside a tuple!

>>> x[2] = 8    # not allowed


# Whole replacement is fine!

 >>> x = (“egg”, 7, 9, 10)
```

Set Data Type

```
>>> s1 = { 1,   2,  3}
>>> s2 = {"ab", "d"}
```

```
>>> s1 = set([1,2,3])
>>> s1
{1, 2, 3}
```

```
>>> s2 = set("Hello")
>>> s2
{'e', 'l', 'o', 'H'}
```

```
>>> s1 = set([1,2,3])
>>> l1 = list(s1)
>>> l1
[1, 2, 3]
>>> l1[0]
1
```

```
>>> s1 = set([1,2,3])
>>> t1 = tuple(s1)
>>> t1
(1, 2, 3)
>>> t1[0]
1
```

Set Operations [1/2]

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])
```

```
>>> s1 & s2
{4, 5, 6}
```

=

```
>>> s1.intersection(s2)
{4, 5, 6}
```

```
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

=

```
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
>>> s1 - s2
{1, 2, 3}
>>> s2 - s1
{8, 9, 7}
```

=

```
>>> s1.difference(s2)
{1, 2, 3}
>>> s2.difference(s1)
{8, 9, 7}
```

Set Operations [2/2]

값 1개 추가하기(add)

이미 만들어진 set 자료형에 값을 추가할 수 있다.
다.

```
>>> s1 = set([1, 2, 3])
>>> s1.add(4)
>>> s1
{1, 2, 3, 4}
```

값 여러 개 추가하기(update)

여러 개의 값을 한꺼번에 추가(update)할 때

```
>>> s1 = set([1, 2, 3])
>>> s1.update([4, 5, 6])
>>> s1
{1, 2, 3, 4, 5, 6}
```

특정 값 제거하기(remove)

특정 값을 제거하고 싶을 때는 아래

```
>>> s1 = set([1, 2, 3])
>>> s1.remove(2)
>>> s1
{1, 3}
```

Dictionary Data Type

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
```

key	value
name	pey
phone	01199993323
birth	1118

```
>>> dic['name']
'pey'
>>> dic['phone']
'0119993323'
>>> dic['birth']
'1118'
```

Insert and Delete in Dictionary

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{2: 'b', 1: 'a'}
```

```
>>> a['name'] = 'pey'
{'name':'pey', 2: 'b', 1: 'a'}
```

```
>>> a[3] = [1,2,3]
{'name': 'pey', 3: [1, 2, 3], 2: 'b', 1: 'a'}
```

```
>>> del a[1]
>>> a
{'name': 'pey', 3: [1, 2, 3], 2: 'b'}
```

Dictionary 만들 때 주의사항

중복되는 Key 값은 금지

```
>>> a = {1:'a', 1:'b'}
>>> a
{1: 'b'}
```

Key 값은 hashable value만 허락

```
>>> a = {[1,2] : 'hi'}
Traceback (most recent call last):
File "", line 1, in ?
TypeError: unhashable type
```

Taxonomy of Python Functions

- Python Built-in Functions:
  - input(), print(), len(), abs(), set(), …..
- User Defined Functions
- Functions belonging to Python Basic Data Types

```
>>> L = [3, 5, 5]
>>> L.append(4)
```

```
>>> S = {3, 5, 6}
>>> S.remove(5)
```

- Functions belonging to User-Defined Classes

```
>>> class Box(object):
        def  calc_space(self):
            ……
```

```
>>> bbb = Box()
>>> space_value = bbb.calc_space()
```

- Functions from Other Modules (consisting of Functions and Classes)

```
>>>  import sam_module
```

- Functions from Python Standard Library (consisting of Functions and Classes)

```
>>>  import math
```

Python Built-in Functions

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Using Libraries (or Modules) [1/3]

- Modules contain a set of useful functions or classes

- Some additional Python standard libraries like “math”, “time” or “datetime” etc. can be imported using keyword “import”

- Python standard libraries are listed here: https://docs.python.org/3/library/

```
# Importing math standard library
>> import math
>> math.sqrt(3)
1.7320508075688772
>> - b + math.sqrt(b * b – 4 * a * c) / ( 2 * a )
```

- help(<module name>) can give you lots of information

Using Libraries (or Modules) [2/3]

- When a Python program starts, it only has access to basic functions and classes

  (len(), sum(), set(), list(), range(), …..)

- Use “import” to tell Python to load a module
  - “import” vs “from ... import ...”

```
>>> # If you want to use cos function in the math library
>>> import math
math.cos( )
>>> from math import cos, pi
cos( )
>>> from math import *
cos( )
```

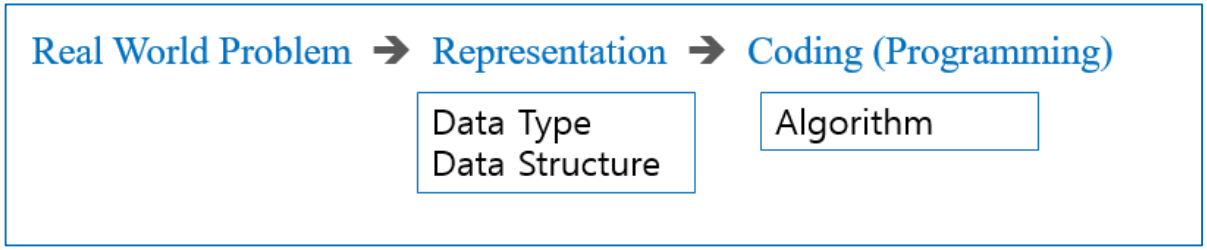
Using Libraries (or Modules) [3/3]

```
>>> import math

>>> math.pi
3.141592653589793
>>> math.cos(0)
1.0
>>> math.cos(math.pi)
-1.0

>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'exp',
 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log',
'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
>>> help(math)
…….
>>> help(math.cos)
……..
```

Temperature Converter

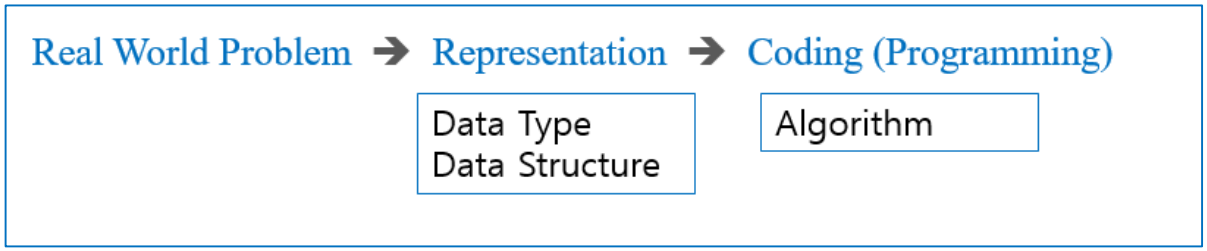
```
>>> def main():
        celsius = eval(input("What is the Celsius temperature? "))
        fahrenheit = (9/5) * celsius + 32
        print("The temperature is ",fahrenheit," degrees Fahrenheit.")
```

```
>>> main()
What is the Celsius temperature? 0
The temperature is  32.0  degrees Fahrenheit.
>>> main()
What is the Celsius temperature? 100
The temperature is  212.0  degrees Fahrenheit.
```



Compute Molecular Weight

```
# Compute molecular weight
# Here are basic weights
#   C → carbon: 12.011 ,   H → hydrogen: 1.0079 ,  O → oxygen: 15.9994
# use round(46.0688, 2) ==> 46.07  #  round() is built-in function


def molecular_wight():
    print("Please enter the number of each atom")
    C = eval(input("carbon: "))
    H = eval(input("hydrogen: "))
    O = eval(input("oxygen: "))
    W = C * 12.011 +  H * 1.0079  +  O * 15.9994
    print ("The molecular weight of C", C, "H", H, "O", O,  "is: ", round(W,2))
```



Factorial Program

```
# factorial.py
#    Program to compute the factorial of a number
#    Illustrates for loop with an accumulator

def main():

    n = eval(input("Please enter a whole number: "))

    fact = 1
    for factor in range(n,1,-1):
       fact = fact * factor

    print("The factorial of", n, "is", fact)
```

```
>>> main()
Please enter a whole number: 100
The factorial of 100 is
 933262154439441526816992388562667004907159682643816214685929
 638952175999932299156089414639761565182862536979208272237582
 511852109168640000000000000000000000
```

Palindrome Checker [1/2]

Palindrome: string을 뒤집어 놓아도 원래와 같은 string
부호와 빈칸을 제외하고 대소문자 구분없이 알파벳이 대칭을 이루는 문장
예를 들어, 'abcdcba'는 뒤집어도 똑같으므로 palindrome
Other Palindrome examples
‘Are we not drawn onward, we few, drawn onward to new era’
‘Do geese see God’
‘Dennis and Edna sinned‘

Step1: User로 부터 string을 받아드린다
Step2: 받아드린 string을 lower case로 바꾼다
Step3: string의 첫번째 character를 마지막 character를 p1, p2로 setting
Step4: p1 < p2인 상태에서는 계속 아래 substep을 수행
Step 3.1: p1과 p2가 가르키는것이 alphabet이 아니면 전진한다
Step 3.2: p1과 p2가 가르키는것이 같은 alphabet이면 Step4로 간다

Palindrome Checker [2/2]

```
def pallindrome_decider():
    P_candidate = input("Type your pallindrome candiate: ")
    print ("Here is your pallindrome candiate:", P_candidate)
    P_candidate = P_candidate.lower()
    print ("After lowering characters ==> ", P_candidate)
    #
    isP_candidate = True
    p1 = 0
    p2 = len(P_candidate) - 1
    #
    while isP_candidate and p1 < p2:
        if P_candidate[p1].isalpha():
            if P_candidate[p2].isalpha():
                if P_candidate[p1]==P_candidate[p2]:
                    p1 = p1 + 1
                    p2 = p2  - 1
                else:   is_candidate = False
            else:  p2 = p2 - 1  # if not alphabet ==> move p2 to left
      else: p1 = p1+1          # if not alphabet ==> move p1 to right
#
    if isP_candidate:
            print ("Yes, your pallindrome candiate", P_candidate,  "is a real pallindrome!")
    else:  print ("No, your pallindrome candiate", P_candidate,  "is not a real pallindrome!")
```

Miscellaneous Things

- Reserved Words
- Multi-operand Comparison
- Simultaneous Assignment
- 2 Dimensional List
- Parameter Forms

Reserved Words

- Some words are part of Python itself.
  - known as **reserved words or keywords**
- This means they are not available for you to use as a name for a vari able, etc. in your program

and	del	for	is	raise
assert	elif	from	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	yield
def	finally	in		

Table 2.1: Python Reserved Words.

Boolean Expressions: Multi-operand comparison

- The following compound comparisons are valid expressions in Python

E.g.

```
>>> 1 < 5 < 7
True

>>> 2 > 1 < 7
True

>>> 5 > 4 > 3.2 >= 1 == 1 != 8
True
```

```
If  0 <= number <= 100 :
        print(number)              # 이것도 가능
```

Elements of Programs: Simultaneous Assignment

- Several values can be calculated at the same time
  - <var>, <var>, … = <expr>, <expr>, …
  - Evaluate the expressions in the RHS and assign them to the variables on the LHS

```
sum, diff = x + y, x - y
```

- How could you use this to swap the values for x and y?
  - Would this work?

    x = y

    y = x
- We could use a temporary variable…
- Or We can swap the values of two variables quite easily in Python!

```
>>> x = 3
>>> y = 4
>>> print (x, y)
3 4
>>> x, y = y, x
>>> print (x, y)
4 3
```

2D List

- List is a basically 1-D Array

```
>>> grades = ['A', 'B', 'C', 'D', 'F']
>>> grades[0]
'A'
>>> grades[2:4]
['C', 'D']
```

Two Dimensional Arrays

- Some data can be organized efficiently in a **table** (also called a **matrix** or **2-dimensional array**)
- Each cell is denoted with two subscripts, a row and column indicator

B[2][3] = 50

B	0	1	2	3	4
0	3	18	43	49	65
1	14	30	32	53	75
2	9	28	38	50	73
3	10	24	37	58	62
4	7	19	40	46	66

2D Lists in Python

```
data = [ [1, 2, 3, 4],
         [5, 6, 7, 8],
         [9, 10, 11, 12]
       ]
```

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
>>> data[0]
[1, 2, 3, 4]
>>> data[1][2]
7
>>> data[2][5]   index error
```

2D List Example in Python

- Find the sum of all elements in a 2D array

```
def sum_matrix(table):
    sum = 0
    for row in range(0,len(table)):
        for col in range(0,len(table[row])):
            sum = sum + table[row][col]
    return sum
```

number of rows in the table

number of columns in the given row of the table

In a rectangular matrix, this number will be fixed so we could use a fixed number for row such as len(table[0])

Tracing the Nested Loop

```
def sum_matrix(table):
    sum = 0
    for row in range(0,len(table)):
        for col in range(0,len(table[row])):
            sum = sum + table[row][col]
    return sum
```

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

`len(table) = 3`

`len(table[row])= 4 for every row`

row	col	sum
0	0	1
0	1	3
0	2	6
0	3	10
1	0	15
1	1	21
1	2	28
1	3	36
2	0	45
2	1	55
2	2	66
2	3	78

Manipulating 2D-Array made by List

Accessing a whole row

```
# alias (not a copy!); cheap (no new list created)
a = [ [ 1, 2, 3 ] , [ 4, 5, 6 ] ]
row = 1
rowList = a[row]
print(rowList)
```

1, 2, 3
4, 5, 6

Accessing a whole column

```
# copy (not an alias!); expensive (new list created)
a = [ [ 1, 2, 3 ] , [ 4, 5, 6 ] ]
col = 1
colList = [ ]
for i in range(len(a)):
    colList += [ a[i][col] ]
print(colList)
```

1, 2, 3
4, 5, 6

Python List is a simple data structure, but not convenient for Matrix

Looking for Better Ways for 2D Array, 3D Array,…..

- Array Module
- NumPy Module

여러 개의 입력값을 받는 함수 [1/2]

- 여러 개의 입력값을 모두 더하는 함수를 만들려 한다.
- 예, sum_many(1, 2) returns 3 sum_many(1,2,3,4,5) returns 15

```
>>> def sum_many(*args):
...     sum = 0
...     for i in args:
...         sum = sum + i
...     return sum
...
>>>
```

여러 개의 입력값이 들어오는것을 기대

```
>>> result = sum_many(1,2,3)
>>> print(result)
6
>>> result = sum_many(1,2,3,4,5,6,7,8,9,10)
>>> print(result)
55
```

List값을 입력값으로 하고 싶다면!!

```
>>> input_data = [1, 2, 3]
>>>
sum_many( *input_data )
>>> 6
```

여러 개의 입력값을 받는 함수 [2/2]

```
>>> def sum_mul(choice, *args):
...     if choice == "sum":
...         result = 0
...         for i in args:
...             result = result + i
...     elif choice == "mul":
...         result = 1
...         for i in args:
...             result = result * i
...     return result
...
>>>
```

```
>>> result = sum_mul('sum', 1,2,3,4,5)
>>> print(result)
15
>>> result = sum_mul('mul', 1,2,3,4,5)
>>> print(result)
120
```

List값을 입력값으로 하고 싶다면!!

```
>>> input_data = [1, 2, 3]
>>> sum_mul( 'sum',   *input_data)
>>> 6
```

여러 개의 Parameter = Value 형태를 받는 함수 [1/2]

```
def sum(*values, **options):
    s = 0
    for i in values:
        s = s + i
    if "neg" in options:
        if options["neg"]:
            s = -s
    return s

s = sum(1, 2, 3, 4, 5)            # returns 15
s = sum(1, 2, 3, 4, 5, neg=True)  # returns -15
s = sum(1, 2, 3, 4, 5, neg=False) # returns 15
```

여러개의 parameter = value 형태로 들어오는것을 기대

```
>>> input_data = [1, 2, 3]
>>> option_data = { neg: True }
>>> sum(*input_data,  **option_data)
>>> -6
```

List값과 Dictionary값을 입력값으로 하고 싶다면!!

여러 개의 Parameter = Value 형태를 받는 함수 [2/2]

여러개의 parameter = value
형태로 들어오는것을 기대

```
>>> def foo(**option):
        if 'b' in option:
            print("Mr. Lee")
        else:
            print("None")

>>> foo(a = 'Kim')
>>> None
>>> foo(a= 'Kim', b = 'Lee')
>>> Mr. Lee
>>>
>>> data = {'a': 'kim', 'b': 'Lee'}
>>> foo(**data)
>>> Mr. Lee
```