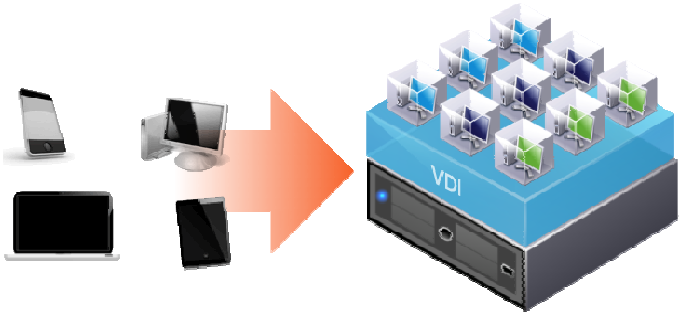
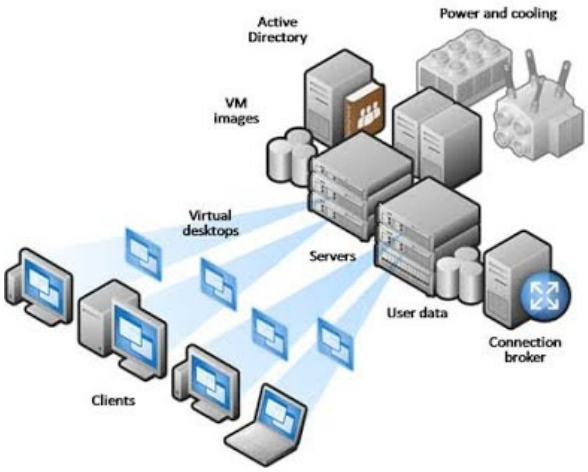
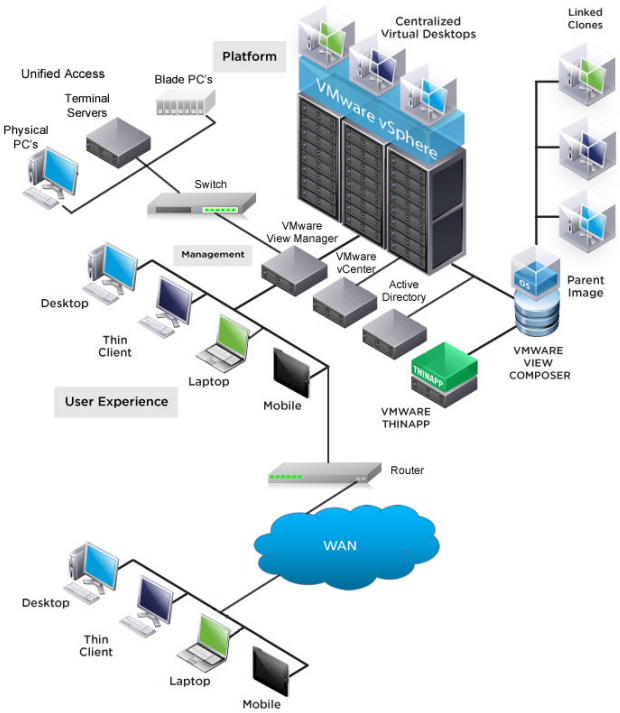
Python Programming

- VDI (Virtual Desktop Interface)란??



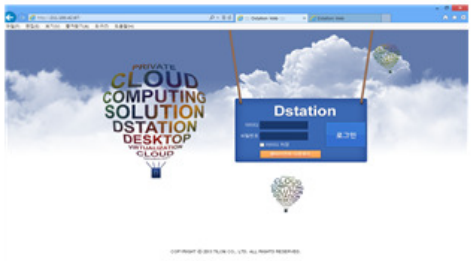




• 서강대 VDI 접속

1. 가상 PC 웹페이지 접속

- Internet Explorer 주소창에 "http://163.239.4.150" 을 입력하여 접속 합니다.



2. 가상 PC 접속프로그램 설치

- 가상 PC 접속프로그램 설치를 진행 합니다.

1) 다운로드를 클릭합니다.

2) 가상 PC 접속 프로그램(Linker Desktop) 다운로드 후 실행하여 설치를 진행합니다.

cyber.sogang.ac.kr : 프로그래밍 언어 ➔ 강의자료 ➔ "가상머신 사용 설명서 "

3. 가상 PC 실행

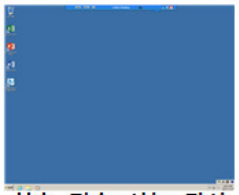
- 인증서버 정보를 입력합니다.(최초 실행 시)

1) 이름 : 접속 서버 정보 (e.g. SGVDI)

2) 서버 URL : "163.239.4.150"

3) 포트 : 80

- ID(학번)/PW(123qwe) 를 입력하고 로그인합니다.
- 할당된 가상 PC를 실행하여 가상 PC로 접속합니다.



※ 최초 접속 시는 가상 PC 환경 구성으로 접속시간이 다소 오래 걸릴 수 있습니다.

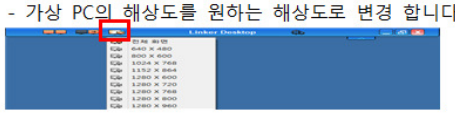
4. 싱글/듀얼 모니터 기능

- 가상 PC의 화면을 모니터 좌/우로 변경 합니다.

- 가상 PC의 화면을 전체 화면으로 변경합니다. (듀얼모니터 시 모니터 화면 전체 사용)

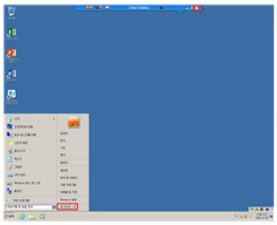
5. 해상도 변경 기능

- 가상 PC의 해상도를 원하는 해상도로 변경 합니다.



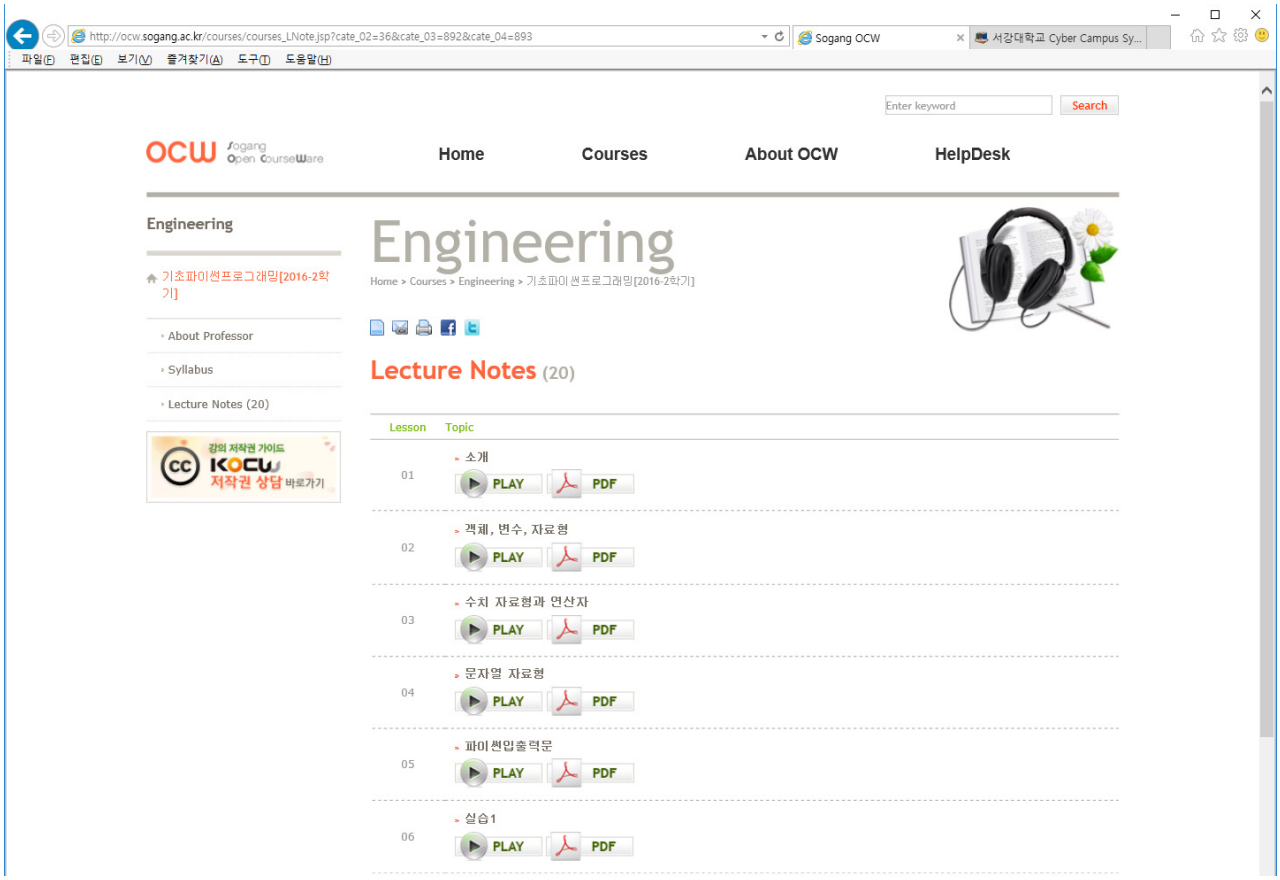
6. 가상PC 종료 / 재시작

- [가상머신 재시작] 가상 PC를 재시작합니다. (재부팅)
- [가상머신 종료] 가상 PC를 종료합니다.

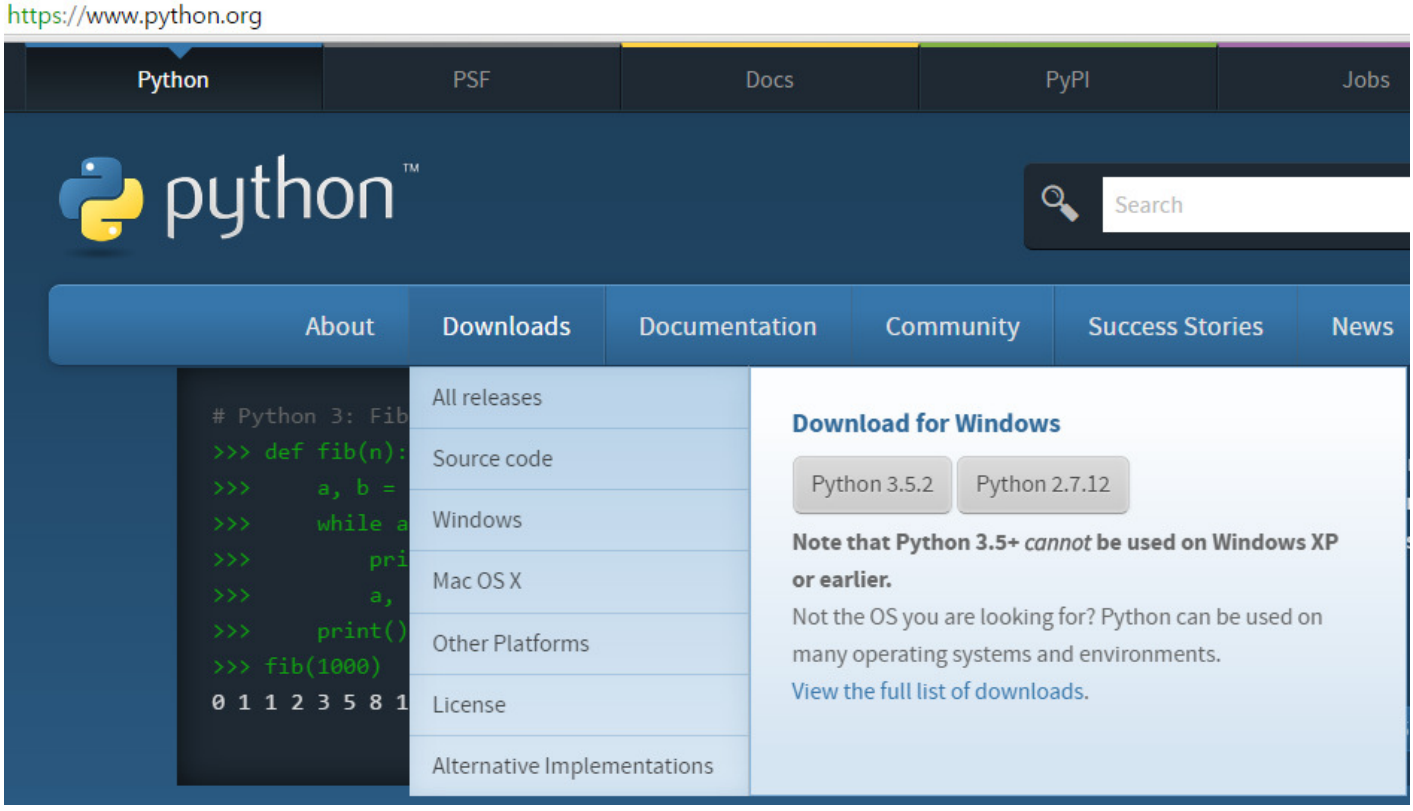


- [닫기] 버튼은 접속 종료이며, 일정시간 동안 작업 화면을 유지합니다.
- [로그오프]는 작업 환경을 종료합니다.

- **Python 강의 자료 : ocw.sogang.ac.kr ➔ 기초파이썬프로그래밍[2016-2학기]**



- **Python Interpreter Download**



연습 문제 1 : 아래 왼쪽과 같이 **ages** 사전에는 이름과 나이가 저장되어 있다 (이름이 '키'이므로 같은 이름이 없다고 가정한다). 오른쪽과 같은 실행 결과를 내는 프로그램을 작성하시오.

```
ages = {}
ages['Alice'] = 23
ages['Paul'] = 25
ages['Peter'] = 19
ages['Karen'] = 40
ages['Andy'] = 25
ages['Cindy'] = 30
ages['David'] = 19
ages['Sally'] = 28
ages['Tom'] = 22
ages['Sue'] = 32
ages['Bob'] = 31
```

```
Enter name : Peter
Peter is 19 years old.

Enter name : Tom
Tom is 22 years old.

Enter name : Beth
Beth is not in the 'age' dictionary.
```

```
ages = {}
ages['Alice'] = 23
ages['Paul'] = 25
ages['Peter'] = 19
ages['Karen'] = 40
ages['Andy'] = 25
ages['Cindy'] = 30
ages['David'] = 19
ages['Sally'] = 28
ages['Tom'] = 22
ages['Sue'] = 32
ages['Bob'] = 31

name = input('Enter name : ')
if name not in ages:
    print(name,"is not in the 'age' dictionary.")
else:
    print(name, 'is' , ages[name], 'years old.')
```


연습 문제 2 : 리스트 **voca**에는 몇 개의 단어가 저장되어 있다 (중복된 단어들이 있다). 각 단어가 **voca**에 몇 번 저장되어 있는지를 출력하는 프로그램을 작성하시오. (반드시 사전을 이용하여야 한다)
voca의 내용이 바뀌면 결과도 따라서 바뀌어야 한다 (아래 예제 참고).

```
voca = ['rose', 'candle', 'paper', 'book', 'song', 'candle',
        'berry', 'paper', 'berry', 'berry', 'song', 'paper',
        'rose', 'song', 'lion', 'berry', 'rose', 'book',
        'rose', 'rose']
```

```
candle - 2 times
rose - 5 times
song - 3 times
berry - 4 times
book - 2 times
paper - 3 times
lion - 1 times
```

```
voca = ['python', 'c', 'c++', 'java', 'java', 'javascript', 'python', 'ruby',
        'swift', 'c', 'c', 'java', 'python', 'ruby', 'swift', 'cobol', 'cobol',
        'r', 'r', 'c', 'c', 'r', 'html', 'html', 'ruby']
```

```
python - 3 times
swift - 2 times
java - 3 times
html - 2 times
r - 3 times
c - 5 times
ruby - 3 times
c++ - 1 times
javascript - 1 tim
cobol - 2 times
```

```
voca = ['python', 'c', 'c++', 'java', 'java', 'javascript', 'python', 'ruby',
        'swift', 'c', 'c', 'java', 'python', 'ruby', 'swift', 'cobol', 'cobol',
        'r', 'r', 'c', 'c', 'r', 'html', 'html', 'ruby']

voca_dict = {}

for word in voca:
    if word in voca_dict:
        voca_dict[word] += 1
    else:
        voca_dict[word] = 1

for w,c in voca_dict.items():
    print(w, '-', c, 'times')
```

연습 문제 3 : 주어진 학생들을 대상으로 몇 개의 팀을 만들려고 한다. 함수 make_teams는 학생 리스트 student와 구성하고자 하는 팀의 개수 count을 입력받아서 student에 있는 학생들을 count개의 팀으로 나누어 반환한다. 팀을 나눌 때는 다음과 같은 방식을 취한다. 예를 들어, 학생 리스트 A에 8명의 학생이 저장되어 있다고 할 때 4팀으로 나누고자 한다면, 학생 리스트에 0,1,2,3,0,1,2,3의 순서로 팀을 구성하여야 하는데 같은 번호에 해당하는 학생이 같은 팀으로 들어가야 한다. main 부분은 수정하지 않고 그대로 가져다 쓰고 두 개의 함수를 완성한다.

```
def make_teams(student, count):
    L = []
    for i in range(count):
        L.append([])
    for team in range(count):
        for i in range(0, len(student), count):
            if (team + i) < len(student):
                L[team].append(student[team+i])
    return L

def print_team(T):
    for i in T:
        print(i)
```

```
# main

A = ['alice', 'paul', 'cindy', 'steve', 'bill', 'tom',
'donald', 'hilary']
B = ['kelly', 'jennifer', 'jessica', 'brad', 'david',
'andy', 'joe', 'joye', 'sally', 'mary', 'stella', 'bob',
'tammy', 'suzy', 'peter', 'sophie', 'oscar']
C = A + B

print('A를 4팀으로 나눈다')
print('-' * 30)
T = make_teams(A, 4)
print_team(T)

print('=' * 30)

print('B를 3팀으로 나눈다')
```

```
A를 4팀으로 나눈다
------------------------------
['alice', 'bill']
['paul', 'tom']
['cindy', 'donald']
['steve', 'hilary']
==============================
B를 3팀으로 나눈다
------------------------------
['kelly', 'brad', 'joe', 'mary', 'tammy', 'sophie']
['jennifer', 'david', 'joye', 'stella', 'suzy', 'oscar']
['jessica', 'andy', 'sally', 'bob', 'peter']
==============================
C를 7팀으로 나눈다
------------------------------
['alice', 'hilary', 'joe', 'suzy']
['paul', 'kelly', 'joye', 'peter']
['cindy', 'jennifer', 'sally', 'sophie']
['steve', 'jessica', 'mary', 'oscar']
['bill', 'brad', 'stella']
['tom', 'david', 'bob']
['donald', 'andy', 'tammy']
```


- **Python 프로그래밍 과제**
  - **cyber.sogang.ac.kr ➔ 프로그래밍 언어 ➔ 파이선프로그래밍 과제.hwp 에 있는 10문제를 프로그래밍하여 제출**
  - **제출방법 :**

    ⇔ "1_학번.py" 등과 같이 각 문제에 대한 파리선 소스를 저장시켜서, 파일 10개를 만들고, 이를 "학번.zip"으로 하여 cyber.sogang.ac.kr➔ 과제로 제출

- **Python**
  - uses a mechanism, which is known as "Call-by-Object", sometimes also called "Call by Object Reference" or "Call by Sharing".
  - If you pass *immutable arguments* like integers, strings or tuples to a function, the passing acts like *call-by-value.*
    - ⇔ The object reference is passed to the function parameters.
    - ⇔ They can't be changed within the function, because they can't be changed at all, i.e. they are immutable.
  - It's different, if we pass mutable arguments. They are also passed by object reference, but they *can be changed* in place in the function.
    - ⇔ If we pass a list to a function, we have to consider two cases:
      - ⇒ Elements of a list can be changed in place, i.e. the list will be changed even in the caller's scope.
      - ⇒ If a new list is assigned to the name, the old list will not be affected, i.e. the list in the caller's scope will remain untouched.

Python initially behaves like call-by-reference, but as soon as we are changing the value of such a variable, Python "switches" to call-by-value.

```
def ref_demo(x):
    print "x=",x," id=",id(x)
    x=42
    print "x=",x," id=",id(x)
```

```
>>> x = 9
>>> id(x)
41902552
>>> ref_demo(x)
x= 9  id= 41902552
x= 42  id= 41903752
>>> id(x)
41902552
>>>
```

```
>>> def func1(list):
...     print list
...     list = [47,11]
...     print list
...
>>> fib = [0,1,1,2,3,5,8]
>>> func1(fib)
[0, 1, 1, 2, 3, 5, 8]
[47, 11]
>>> print fib
[0, 1, 1, 2, 3, 5, 8]
>>>
```

```
>>> def func2(list):
...     print list
...     list += [47,11]
...     print list
...
>>> fib = [0,1,1,2,3,5,8]
>>> func2(fib)
[0, 1, 1, 2, 3, 5, 8]
[0, 1, 1, 2, 3, 5, 8, 47, 11]
>>> print fib
[0, 1, 1, 2, 3, 5, 8, 47, 11]
>>>
```


  - It is important to understand that *variables in Python are really just references to objects in memory*
  - containers and user-defined types are generally mutable while everything else is immutable.
    - ⇔ immutable objects
      - ⇒ an immutable object (unchangeable object) is an object whose state cannot be modified after it is created
      - ⇒ Numeric types (int, float, complex), string, tuple, frozen set, bytes
    - ⇔ mutable objects : list, dict, set, byte array

only mutable objects can be changed in place (l[0] = 1 is ok in our example, but s[0] = 'a' raises an error).

```
a = 1
s = 'abc'
l = ['a string', 456, ('a', 'tuple', 'inside', 'a', 'list')]

/* create new objects, and the variable point to a
different object (newly created ones in our
examples).  */
a = 7
s = 'xyz'
l = ['a simpler list', 99, 10]
```

```
def append_to_sequence (myseq):
    myseq += (9,9,9)
    return myseq

tuple1 = (1,2,3)      # tuples are immutable
list1  = [1,2,3]      # lists are mutable

tuple2 = append_to_sequence(tuple1)
list2  = append_to_sequence(list1)

print 'tuple1 = ', tuple1  # outputs (1, 2, 3)
print 'tuple2 = ', tuple2  # outputs (1, 2, 3, 9, 9, 9)
print 'list1  = ', list1   # outputs [1, 2, 3, 9, 9, 9]
print 'list2  = ', list2   # outputs [1, 2, 3, 9, 9, 9]
```

  - When used on an immutable object (as in a += 1 or in s += 'qwertz'), Python will silently create a new object and make the variable point to it.
  - However, when used on a mutable object (as in l += [1,2,3]), the object pointed to by the variable will be changed in place.

– Command Line Arguments

```
# Module sys has to be imported:
import sys

# Iteration over all arguments:
for eachArg in sys.argv:
        print eachArg
```

argumente.py

```
>>> python argumente.py python course for beginners

argumente.py
python
course
For
beginners
```

– Variable Length Arguments

⇔ The asterisk "*" is used in Python to define a variable number of arguments. The asterisk character has to precede a variable identifier in the parameter list.

```
>>> def varpafu(*x): print(x)
...
>>> varpafu()
()
>>> varpafu(34,"Do you like Python?", "Of course")
(34, 'Do you like Python?', 'Of course')
>>>
```

```
def arithmetic_mean(x, *l):
    """ The function calculates the arithmetic mean of a
        non-empty arbitrary number of numbers """
    sum = x
    for i in l:
        sum += i

    return sum / (1.0 + len(l))
```

- **Homework**
  - ✓ Python built-in data types and parameter passing method (2 pages)

```
>>> from statistics import arithmetic_mean
>>> arithmetic_mean(4,7,9)
6.666666666666667
>>> arithmetic_mean(4,7,9,45,-3.7,99)
26.71666666666667
```