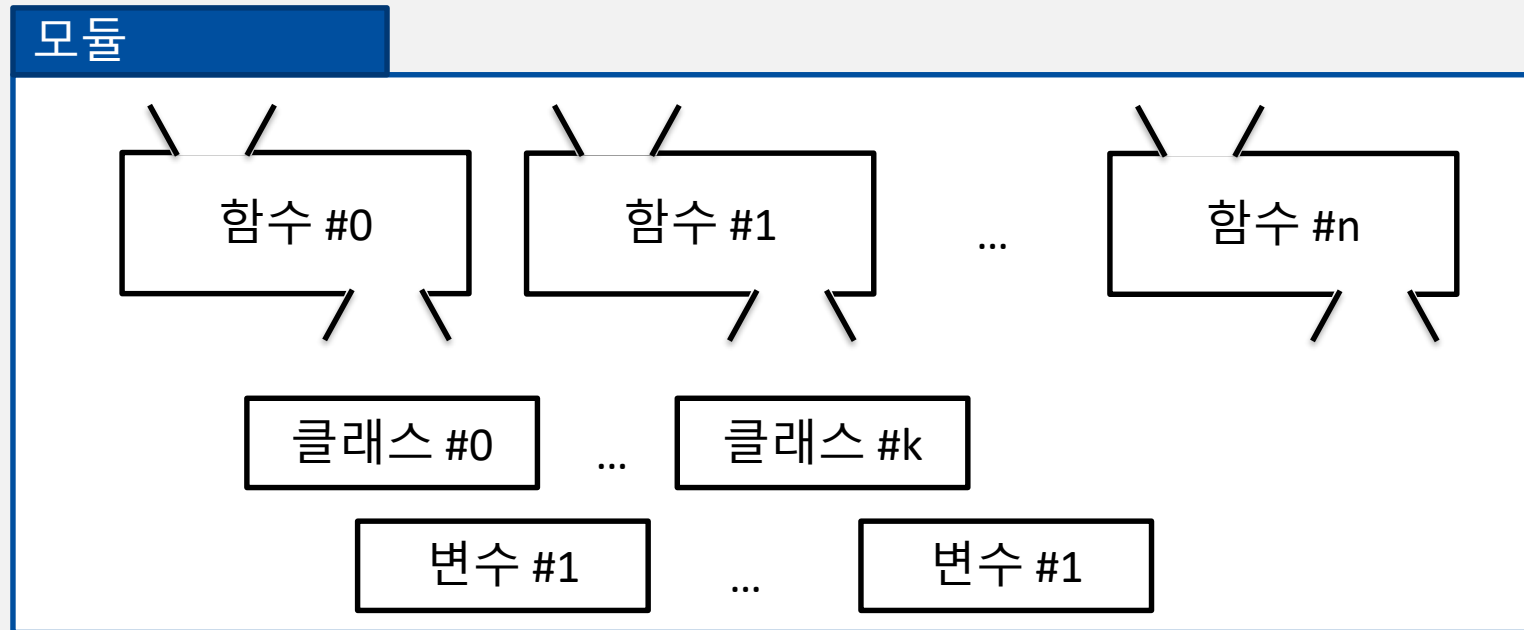


Python Module (I)

김준호 / 박수현

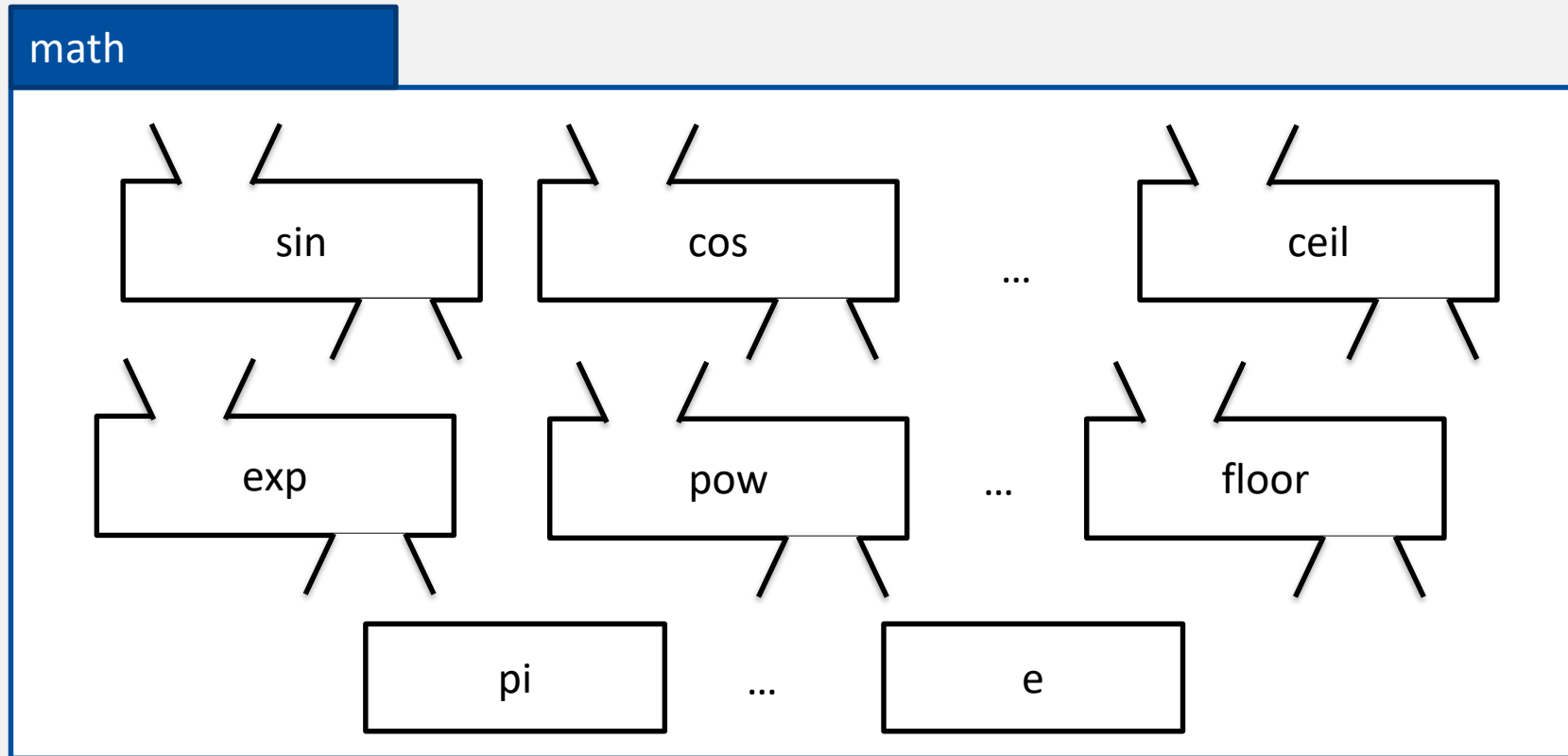
모듈(module)이란?

- Python 함수와 객체들의 묶음을 모듈(module)이라 함
 - 커다란 program을 구성하는 작은 조각
 - 커다란 program을 여러 개의 모듈 또는 file로 나눌 수 있음
 - Computer 저장공간 내에 별도의 file로 구성



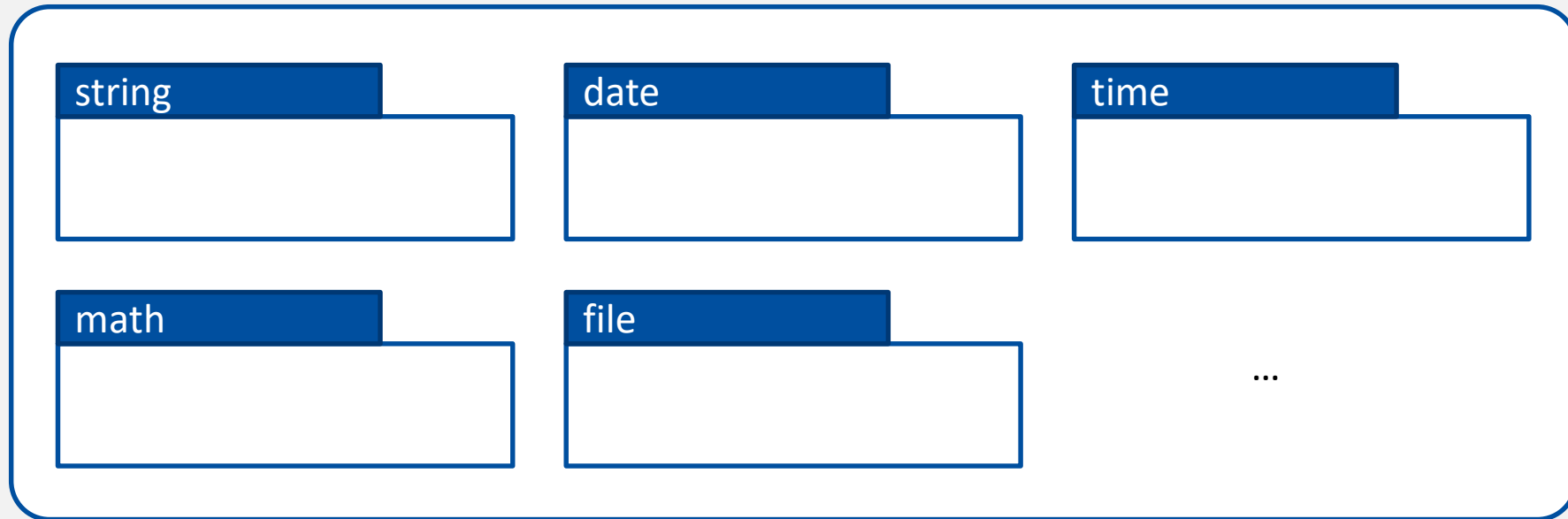
모듈의 예: 수학(math) 모듈

- 예를 들어, 아래 그림과 같이 각종 수학 함수들과 상수들을 묶어 수학(math) 모듈로 정의할 수 있음



Python 빌트인 모듈 (built-in module)

- Python 언어는 programming에서 공통적으로 많이 쓰일 수 있는 문자열(**string**), 날짜(**date**), 시간(**time**), 수학(**math**), 파일(**file**) 등 **200여개 이상의 모듈을 빌트인 모듈*** 형태로 가지고 있음



[*빌트인 모듈 (built-in module): 이미 만들어져 Python에 들어있는 모듈]

Python 빌트인 모듈 (built-in module) 예)

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()	

모듈(module)사용 이유

- Module로 만들어 두면 여러 program에서 해당 모듈을 사용할 수 있음
- 해당하는 code를 찾기가 쉬워 짐
- 서로 다른 작업에 각기 다른 부분들을 조합해서 사용할 수 있음
 - 마치 같은 레고 블록들로 서로 다른 것들을 만들어낼 수 있음

```

1 # Using a module
2 # my_module contains the c_to_f(), f_to_c() function
3
4 import my_module
5
6 print(">> 화씨온도를 섭씨로 변경하고자 하면 0을 하세요.")
7 print("    섭씨온도를 화씨로 변경하고자 하면 1을 하세요.")
8
9 flag = int(input(">> 0 or 1을 입력하세요 : "))
10 print(">> flag :", flag)
11
12 if flag == 1 : # c to f
13
14     celsius = float(input("\n>>> 섭씨 온도를 입력하세요 : "))
15     print(">> 입력 값 " + str(celsius) + " in Celsius")
16
17     fahrenheit = my_module.c_to_f(celsius)
18     print(str(celsius) + " in Celsius is " + str(fahrenheit) + " degrees in Fahrenheit")
19
20 elif flag == 0: # f to c
21     fahrenheit = float(input("\n>>> 화씨 온도를 입력하세요 : "))
22     print(">> 입력 값 " + str(fahrenheit) + " in Fahrenheit")
23
24     celsius = my_module.f_to_c(fahrenheit)
25     print(str(fahrenheit) + " in Fahrenheit is " + str(celsius) + " degrees in Celsius")
26
27 else:
28     print(">> 잘못된 입력입니다. 0 or 1을 입력하세요.")
29

```

사용자 정의 모듈(module) 예제

```

1 # Module
2
3 def c_to_f(celsius):
4     fahrenheit = celsius * 9.0 / 5 + 32
5     return fahrenheit
6
7
8 def f_to_c(fahrenheit):
9     celsius = (fahrenheit - 32) * (5.0 / 9.0)
10    return celsius
11

```

```

>> 화씨온도를 섭씨로 변경하고자 하면 0을 하세요.
    섭씨온도를 화씨로 변경하고자 하면 1을 하세요.
>> 0 or 1을 입력하세요 : 0
>> flag : 0

```

```

>>> 화씨 온도를 입력하세요 : 32
>> 입력 값 32.0 in Fahrenheit
32.0 in Fahrenheit is 0.0 degrees in Celsius
>>>
>>>

```

===== RESTART: C:\W과소사\W과소사-강의

```

>> 화씨온도를 섭씨로 변경하고자 하면 0을 하세요.
    섭씨온도를 화씨로 변경하고자 하면 1을 하세요.
>> 0 or 1을 입력하세요 : 1
>> flag : 1

```

```

>>> 섭씨 온도를 입력하세요 : 0
>> 입력 값 0.0 in Celsius
0.0 in Celsius is 32.0 degrees in Fahrenheit
>>>
>>>

```

===== RESTART: C:\W과소사\W과소사-강의

```

>> 화씨온도를 섭씨로 변경하고자 하면 0을 하세요.
    섭씨온도를 화씨로 변경하고자 하면 1을 하세요.
>> 0 or 1을 입력하세요 : 3
>> flag : 3
>> 잘못된 입력입니다. 0 or 1을 입력하세요.

```

모듈 사용하기

- 모듈 사용하기
빌트인 모듈을 사용해 보자!
- 모듈 내용 들여다 보기
어떤 함수와 변수들이 모듈에 들어있나?
- 모듈의 실체
모듈은 실제로 computer 안에서 어떤 형태로 저장되어 있고 어디에 있나?

모듈 사용해 보기 (I)

Module import

- import문을 사용하여 특정 모듈을 program에 포함
- Name space를 통해 모듈에 포함된 상수(혹은 변수) 사용
- Name space를 통해 모듈에 포함된 함수 사용

Python codes

```
import math
```

```
# math 모듈에 포함된 상수 활용  
print("PI = {0}".format(math.pi))  
print("자연상수 e = {0}".format(math.e))
```

```
# math 모듈에 포함된 sin/cos 함수 활용  
val = math.sin(math.pi / 2)  
print("sin(PI/2) = {0}".format(val))  
val = math.cos(math.pi / 2)  
print("cos(PI/2) = {0}".format(val))
```

모듈 사용해 보기 (I)

모듈에 포함된 상수(변수) 사용

- import문을 사용하여 특정 모듈을 program에 포함
- Name space를 통해 모듈에 포함된 상수(혹은 변수) 사용
- Name space를 통해 모듈에 포함된 함수 사용

Python codes

```
import math

# math 모듈에 포함된 상수 활용
print("PI = {0}".format(math.pi))
print("자연상수 e = {0}".format(math.e))

# math 모듈에 포함된 sin/cos 함수 활용
val = math.sin(math.pi / 2)
print("sin(PI/2) = {0}".format(val))
val = math.cos(math.pi / 2)
print("cos(PI/2) = {0}".format(val))
```

모듈 사용해 보기 (I)

모듈에 포함된 함수 사용

- import문을 사용하여 특정 모듈을 program에 포함
- Name space를 통해 모듈에 포함된 상수(혹은 변수) 사용
- Name space를 통해 모듈에 포함된 함수 사용

Python codes

```
import math
```

```
# math 모듈에 포함된 상수 활용
```

```
print("PI = {0}".format(math.pi))
```

```
print("자연상수 e = {0}".format(math.e))
```

```
# math 모듈에 포함된 sin/cos 함수 활용
```

```
val = math.sin(math.pi / 2)
```

```
print("sin(PI/2) = {0}".format(val))
```

```
val = math.cos(math.pi / 2)
```

```
print("cos(PI/2) = {0}".format(val))
```

모듈 사용해 보기 (I)

모듈 사용법 (종합)

- import문을 사용하여 특정 모듈을 program에 포함
- Name space를 통해 모듈에 포함된 상수(혹은 변수) 사용
- Name space를 통해 모듈에 포함된 함수 사용

Python codes

```
import math
```

```
# math 모듈에 포함된 상수 활용
```

```
print("PI = {0}".format(math.pi))
```

```
print("자연상수 e = {0}".format(math.e))
```

```
# math 모듈에 포함된 sin/cos 함수 활용
```

```
val = math.sin(math.pi / 2)
```

```
print("sin(PI/2) = {0}".format(val))
```

```
val = math.cos(math.pi / 2)
```

```
print("cos(PI/2) = {0}".format(val))
```



math.py

File Edit Format Run Options Window Help

```
1 import math
2
3 # math 모듈에 포함된 상수 활용
4 print("\n>> PI = {0}".format(math.pi))
5 print(">> 자연상수 e = {0}".format(math.e))
6
7 # math 모듈에 포함된 sin/cos 함수 활용
8 val = math.sin(math.pi / 2)
9 print(">> sin(PI/2) = {0}".format(val))
10
11 val = math.cos(math.pi / 2)
12 print(">> cos(PI/2) = {0}".format(val))
13
```

```
>> PI = 3.141592653589793
>> 자연상수 e = 2.718281828459045
>> sin(PI/2) = 1.0
>> cos(PI/2) = 6.123233995736766e-17
>>>
```

모듈 이름이 너무 길어 적기 귀찮아! - alias

기본적인 모듈 사용법

```
import math
```

```
# math 모듈에 포함된 상수 활용
```

```
print("PI = {0}".format(math.pi))  
print("자연상수 e = {0}".format(math.e))
```

```
# math 모듈에 포함된 sin/cos 함수 활용
```

```
val = math.sin(math.pi / 2)  
print("sin(PI/2) = {0}".format(val))  
val = math.cos(math.pi / 2)  
print("cos(PI/2) = {0}".format(val))
```

모듈에 별칭(alias)을 붙여 사용

```
import math as m
```

```
# 별칭 m으로 math 모듈의 상수 활용
```

```
print("PI = {0}".format(m.pi))  
print("자연상수 e = {0}".format(m.e))
```

```
# 별칭 m으로 math 모듈의 함수 활용
```

```
val = m.sin(m.pi / 2)  
print("sin(PI/2) = {0}".format(val))  
val = m.cos(m.pi / 2)  
print("cos(PI/2) = {0}".format(val))
```

```

1 import math as m
2
3 print("\n-- 별칭 m으로 math 모듈 활용")
4
5 # 별칭 m으로 math 모듈의 상수 활용
6 print("\n>> PI = {0}".format(m.pi))
7 print(">> 자연상수 e = {0}".format(m.e))
8
9 # 별칭 m으로 math 모듈의 함수 활용
10 val = m.sin(m.pi / 2)
11 print(">> sin(PI/2) = {0}".format(val))
12 val = m.cos(m.pi / 2)
13 print(">> cos(PI/2) = {0}\n".format(val))
14

```

-- 별칭 m으로 math 모듈 활용

```

>> PI = 3.141592653589793
>> 자연상수 e = 2.718281828459045
>> sin(PI/2) = 1.0
>> cos(PI/2) = 6.123233995736766e-17

```

별칭 마저도 적기 귀찮아!

기본적인 모듈 사용법

```
import math
```

math 모듈에 포함된 상수 활용

```
print("PI = {0}".format(math.pi))  
print("자연상수 e = {0}".format(math.e))
```

math 모듈에 포함된 sin/cos 함수 활용

```
val = math.sin(math.pi / 2)  
print("sin(PI/2) = {0}".format(val))  
val = math.cos(math.pi / 2)  
print("cos(PI/2) = {0}".format(val))
```

현재 name space로 모듈 내용 모두 가져오기

```
from math import *
```

math 모듈의 상수를 현재 네임스페이스로 활용

```
print("PI = {0}".format(pi))  
print("자연상수 e = {0}".format(e))
```

math 모듈의 함수를 현재 네임스페이스로 활용

```
val = sin(pi / 2)  
print("sin(PI/2) = {0}".format(val))  
val = cos(pi / 2)  
print("cos(PI/2) = {0}".format(val))
```



```
1 from math import *
2
3 print("\n-- 별칭 없이 math 모듈 활용 : wild card * 사용")
4
5 # math 모듈의 상수를 현재 네임스페이스로 활용
6 print("\n>> PI = {0}".format(pi))
7 print(">> 자연상수 e = {0}".format(e))
8
9 # math 모듈의 함수를 현재 네임스페이스로 활용
10 val = sin(pi / 2)
11 print(">> sin(PI/2) = {0}".format(val))
12 val = cos(pi / 2)
13 print(">> cos(PI/2) = {0}\n".format(val))
```

-- 별칭 없이 math 모듈 활용 : wild card * 사용

```
>> PI = 3.141592653589793
>> 자연상수 e = 2.718281828459045
>> sin(PI/2) = 1.0
>> cos(PI/2) = 6.123233995736766e-17
```

웬만하면 모듈 이름(혹은 별칭)은 꼭 사용하자

from 000 import * 사용 예

```
e = "다섯번째 알파벳"  
print(e)
```

```
from math import *
```

```
# 의도치 않게 기존 변수 e의 내용이 바뀜  
# 기존의 변수 e가 math.e로 덮어쓰여짐  
print(e)
```

귀찮음이 초래한 예상 못한 결과

```
다섯번째 알파벳  
2.718281828459045
```

웬만하면 모듈 이름(혹은 별칭)은 꼭 사용하자

```
math_alias2.py
File Edit Format Run Options Window Help
1 e = "다섯번째 알파벳"
2 print("\n1) 변수로 선언된 e 값 : ", e)
3
4
5 from math import *
6
7 # 의도치 않게 기존 변수 e의 내용이 바뀜
8 # 기존의 변수 e가 math.e로 덮어씌워짐
9 print("\n2) W"from math import * W" 후의 e 값 : ", e)
10
```

```
1) 변수로 선언된 e 값 : 다섯번째 알파벳
2) "from math import * " 후의 e 값 : 2.718281828459045
...
```

모듈 내용 들여다 보기

- 모듈 안에는 어떤 함수들과 변수들이 있을까?

```
import math
```

```
# math 모듈에 포함된 함수/변수 확인
```

```
print(dir(math))
```



```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',  
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',  
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isfinite', 'isinf',  
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'pi', 'pow', 'radians', 'sin',  
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

모듈 내용 들여다 보기

- 모듈 안에는 어떤 함수들과 변수들이 있을까?
 - 빌트인 함수 혹은 method (builtin_function or method)
 - 실수형 (float)
 - 문자열 (str)
 - 변수형 (type)

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',  
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',  
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isfinite', 'isinf',  
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'pi', 'pow', 'radians', 'sin',  
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

모듈 내용 들여다 보기

- 모듈 안에는 어떤 함수들과 변수들이 있을까?
 - 빌트인 함수 혹은 메서드 (builtin_function_or_method)
 - 실수형 (float)
 - 문자열 (str)
 - 변수형 (type)

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',  
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',  
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isfinite', 'isinf',  
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'pi', 'pow', 'radians', 'sin',  
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

모듈 내용 들여다 보기

- 모듈 안에는 어떤 함수들과 변수들이 있을까?
 - 빌트인 함수 혹은 메서드 (builtin_function_or_method)
 - 실수형 (float)
 - 문자열 (str)
 - 변수형 (type)

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',  
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',  
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isfinite', 'isinf',  
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'pi', 'pow', 'radians', 'sin',  
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

모듈 내용 들여다 보기

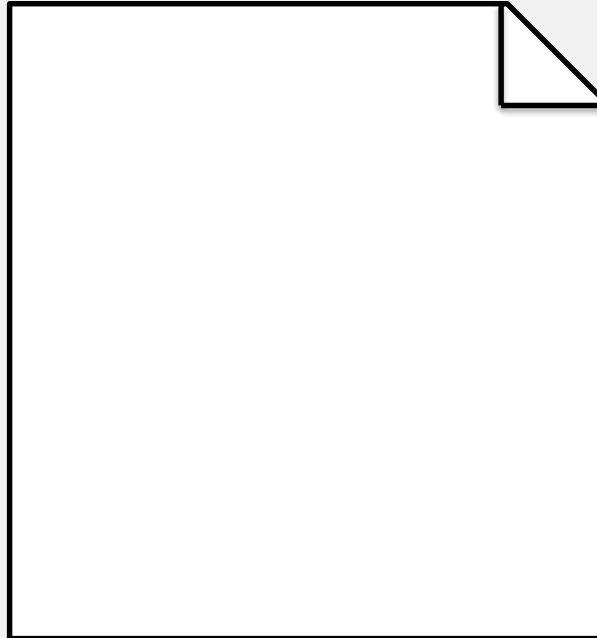
- 모듈 안에는 어떤 함수들과 변수들이 있을까?
 - 빌트인 함수 혹은 메서드 (builtin_function_or_method)
 - 실수형 (float)
 - 문자열 (str)
 - 변수형 (type)

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',  
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',  
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isfinite', 'isinf',  
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'pi', 'pow', 'radians', 'sin',  
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```


모듈의 실체

- 모듈은 Python 코드를 갖고 있는 텍스트 파일 (*.py)
 - my_vector 모듈 == my_vector.py 파일

my_vector.py



모듈의 실체

```
inspect.py
File Edit Format Run Options Window Help
1 import os
2 import inspect
3 import math
4
5 print("\n1)")
6 print(inspect.getfile(os))
7
8 print("\n2)")
9 print(inspect.getfile(inspect))
10
11 print("\n3)")
12 print(os.path.dirname(inspect.getfile(inspect)))
13
14 print("\n4)")
15 print(math)
16
17 print("\n5)")
18 print(inspect)
```

- 우리가 썼던 빌트인 모듈은 컴퓨터 어디에 있나요?
 - Python 프로그래밍으로 알아내기

```
1)
C:\Users\User\AppData\Local\Programs\Python\Python38-32\Lib\os.py
2)
C:\Users\User\AppData\Local\Programs\Python\Python38-32\Lib\inspect.py
3)
C:\Users\User\AppData\Local\Programs\Python\Python38-32\Lib
4)
<module 'math' (built-in)>
5)
<module 'inspect' from 'C:\Users\User\AppData\Local\Programs\Python\Python38-32\Lib\inspect.py'>
...

```

재미있는 모듈 활용의 예

- Computer에게 명령 내리기
subprocess 모듈 활용하기
- 인터넷 접근하기
urllib 모듈 활용하기
- Computer file system 접근하기
os 모듈 활용하기

컴퓨터에게 명령 내리기 – webbrowser 모듈 활용

- 웹 브라우저를 띄워 보자

open-browser.py

File Edit Format Run Options Window Help

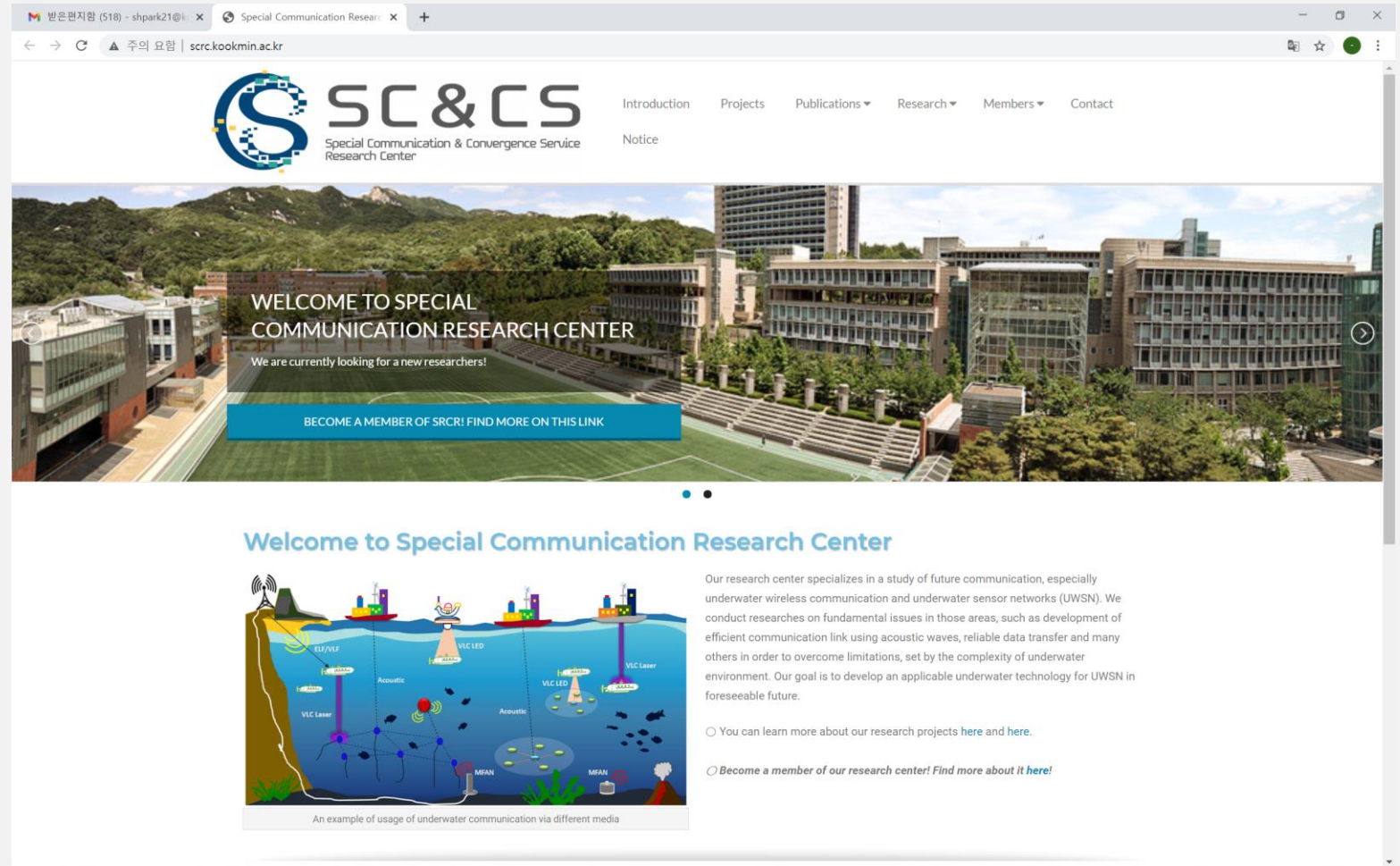
```
import webbrowser
```

```
# 웹 주소 설정
```

```
url = 'http://scrc.kookmin.ac.kr'
```

```
# 기본 웹브라우저 구동
```

```
webbrowser.open(url)
```



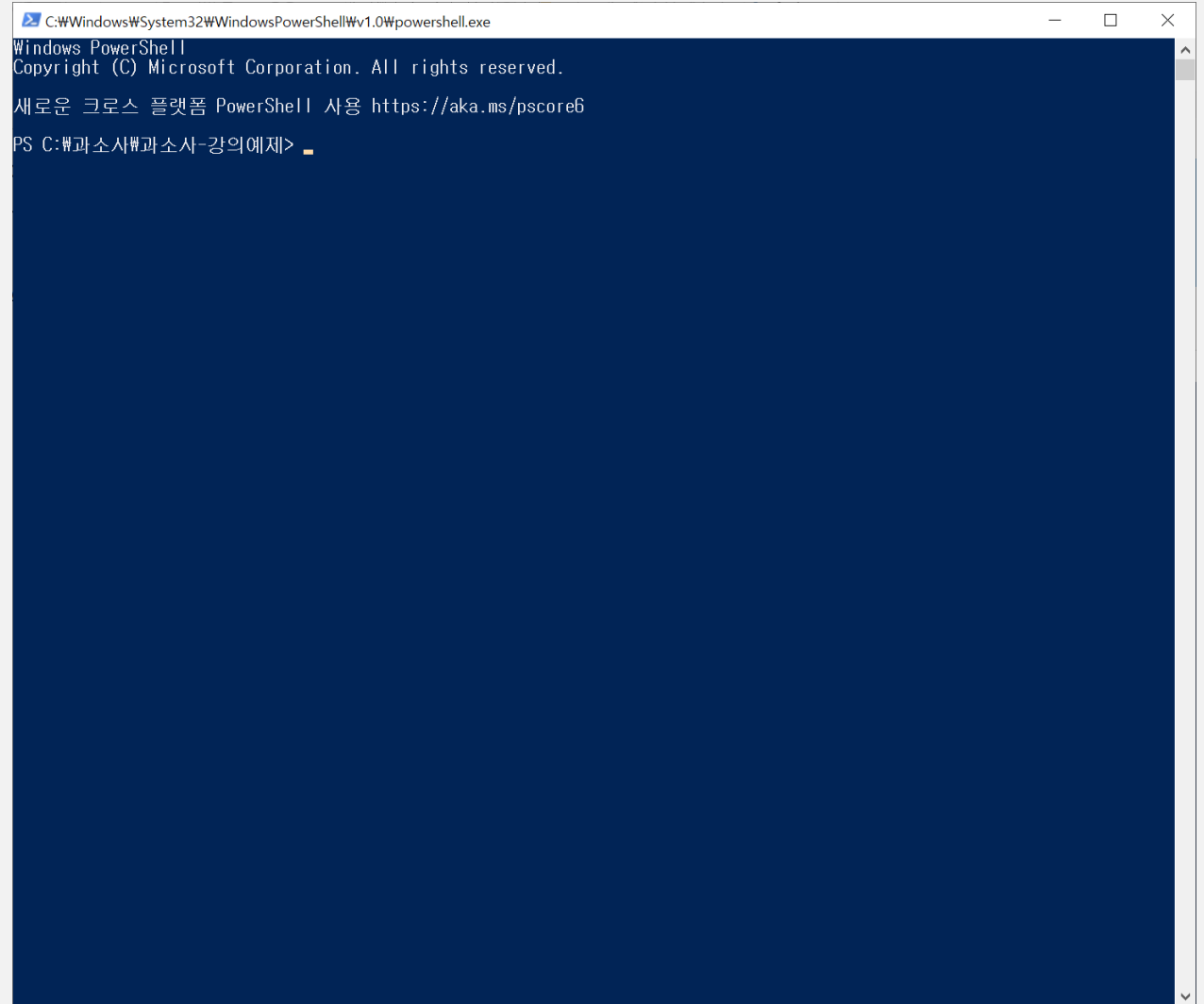
컴퓨터에게 명령 내리기 – subprocess 모듈 활용

- 특정 program 실행하기

open-cmd.py

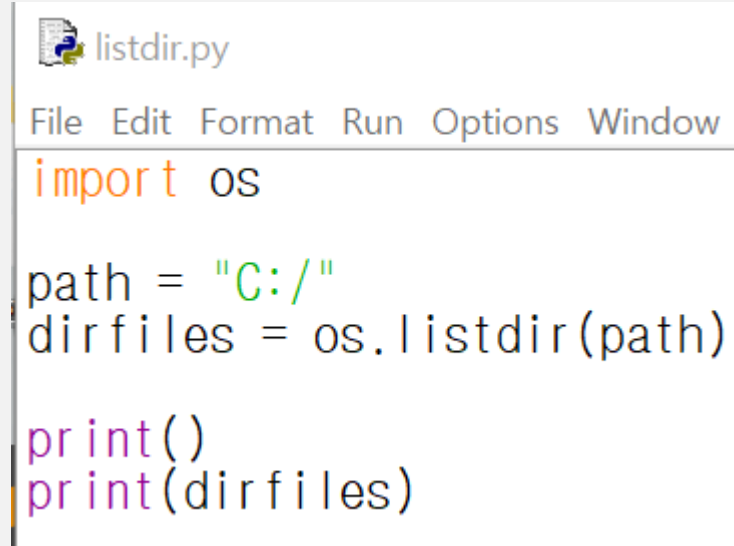
File Edit Format Run Options Window Help

```
1 import subprocess
2
3 # 명령어창 실행 (Windows 운영체제)
4 #subprocess.call("cmd")
5 subprocess.call("powershell")
6
```



컴퓨터 파일 시스템 접근하기 – os 모듈 활용

- 특정 디렉터리 안에는 어떤 디렉터리와 파일들이 있을까?



```
listdir.py
File Edit Format Run Options Window
import os

path = "C:/"
dirfiles = os.listdir(path)

print()
print(dirfiles)
```



```
['$Recycle.Bin', 'Config.Msi', 'Documents and Settings', 'Fuji Xerox', 'hiberfil  
.sys', 'Intel', 'MSOCache', 'pagefile.sys', 'PerfLogs', 'Program Files', 'Progra  
m Files (x86)', 'ProgramData', 'Recovery', 'swapfile.sys', 'System Volume Inform  
ation', 'Temp', 'Users', 'webserver', 'Windows', 'XecureSSL']
```

컴퓨터 파일 시스템 접근하기 – os 모듈 활용

- 디렉터리와 파일을 따로 분리해서 저장하려면?

```
import os

path = "C:/"
dirfiles = os.listdir(path)
dir_names = []
file_names = []

for each in dirfiles:
    full_name = path + each
    if os.path.isdir(full_name):
        dir_names.append(full_name + "/")
    else:
        file_names.append(full_name)

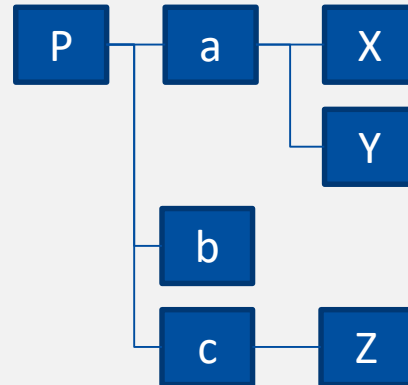
print("dir names: ")
print(dir_names)
print("file names: ")
print(file_names)
```

실습 – 모든 하위 디렉터리 검색

- 특정 디렉터리(폴더)의 모든 하위 디렉터리 찾기
- 예) C:/Python34/ 아래의 모든 하위 디렉터리
 - C:/Python34/
 - C:/Python34/DLLs/
 - C:/Python34/Docs/
 - C:/Python34/Lib/
 - ...
 - C:/Python34/Lib/__pycache__/
 - C:/Python34/Lib/asyncio/
 - ...

실습 – 모든 하위 디렉터리 검색

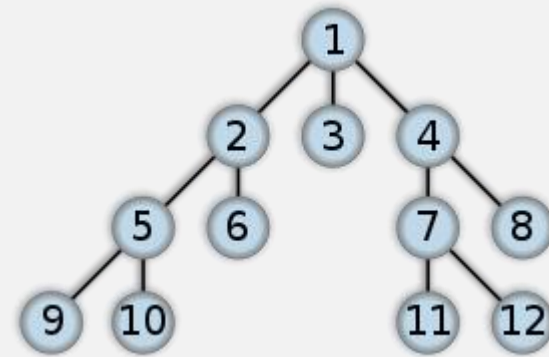
- 다음과 같은 디렉터리 구조인 경우
- 원하는 결과
 - P/
 - P/a/
 - P/b/
 - P/c/
 - P/a/X/
 - P/a/Y/
 - P/c/Z/



실습 – 모든 하위 디렉터리 검색

- 기본 아이디어
 - 모든 하위 디렉터를 저장할 리스트(list)를 하나 만들
 - 우리가 원하는 최종 결과가 저장될 공간
 - 줄 세우기 위한 큐(queue)를 하나 만들
 - 절차수행(알고리즘)을 도와줄 자료구조 클래스
 - 특정 디렉터리(예, C:/Python34/)를 큐에 넣어 줄 세움
 - 맨 앞에 줄 서 있는 디렉터를 큐에서 빼냄
 - 빼낸 디렉터리는 list에 추가(append)
 - 빼낸 디렉터리의 바로 아래 있는 하위 디렉터리들만 찾음
 - 앞서 배운 내용을 응용, 한 디렉터리 바로 아래에 있는 하위 디렉터리들을 찾는 함수 디자인
 - 찾은 하위 디렉터리들을 큐에 줄 세움
 - 이 과정을 큐가 비어있을 때까지 수행

(* 이러한 방식을 breadth-first search라고 함)



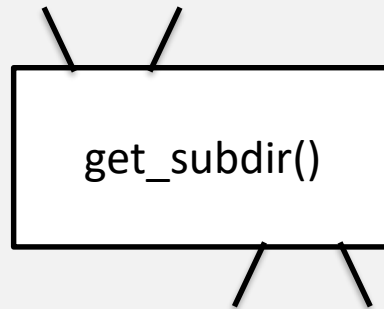
[출처: Wikipedia, [Breadth-first search](#)]

실습 - 모든 하위 디렉터리 검색

- 손으로 돌려보기

줄 서는 방향
→
줄 (queue)

입력 디렉터리의 바로
아래 하위 디렉터리들을
구하는 함수

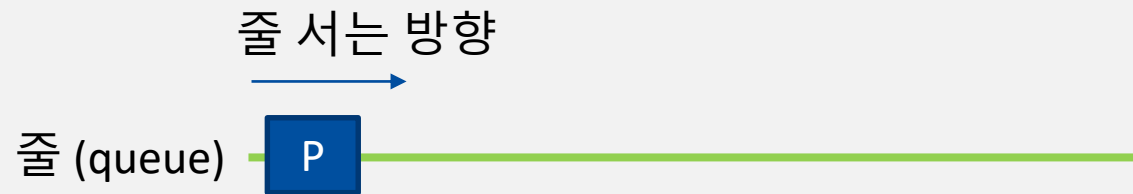


결과 저장소
(list)

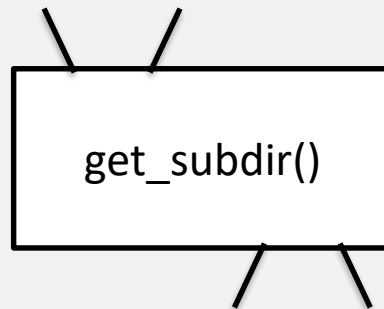


실습 – 모든 하위 디렉터리 검색

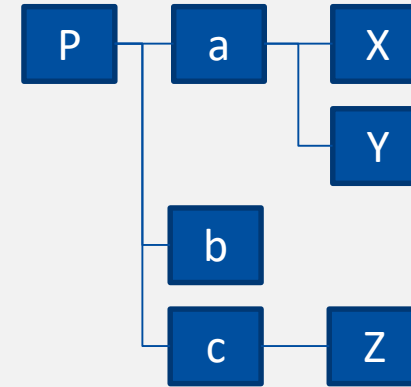
- 손으로 돌려보기
 - 특정 디렉터리(P)를 큐에 넣어 줄 세움



입력 디렉터리의 바로
아래 하위 디렉터리들을
구하는 함수

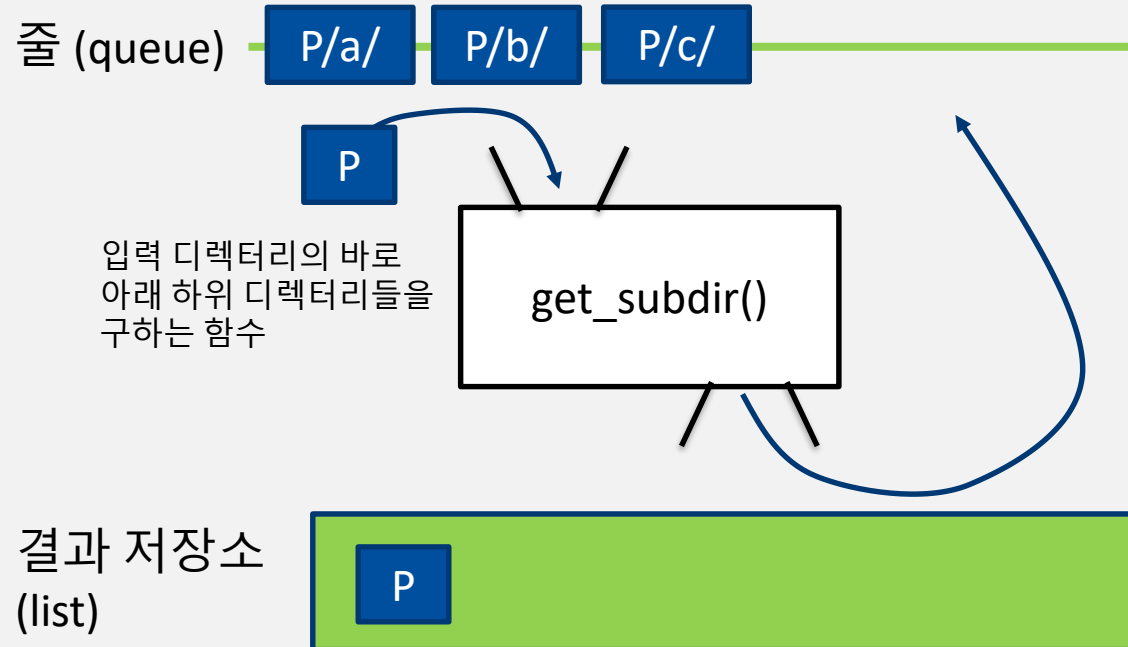
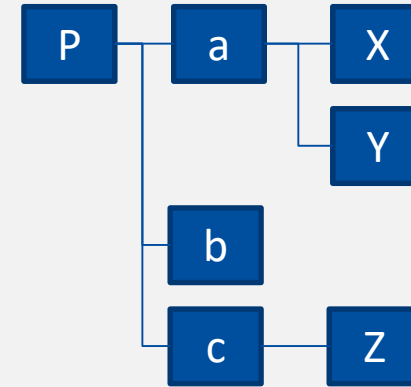


결과 저장소
(list)



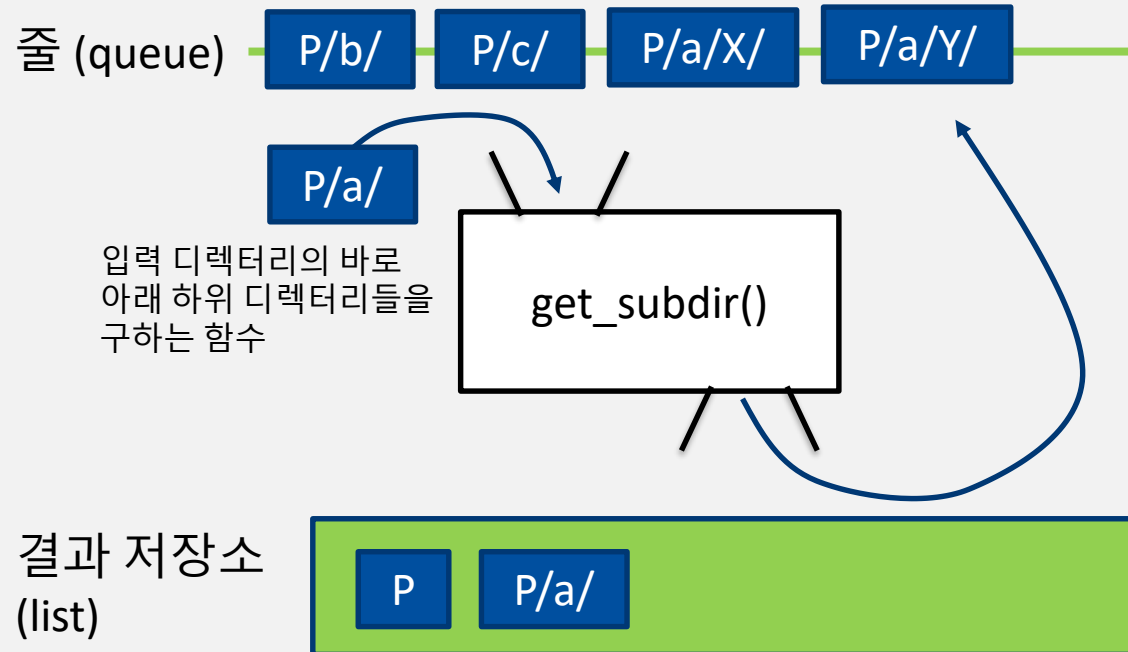
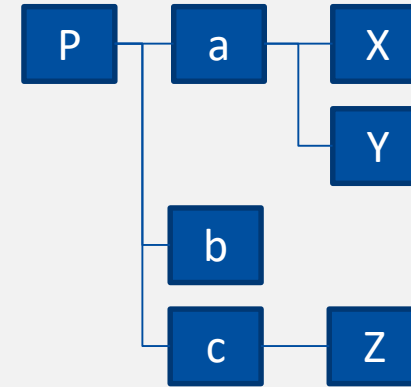
실습 – 모든 하위 디렉터리 검색

- 손으로 돌려보기
 - 줄 맨 앞 디렉터리를 빼냄
 - 빼낸 디렉터리에 대한 바로 아래 하위 디렉터리들을 구해 줄 세움
 - 빼낸 디렉터리는 결과 저장소(list)에 저장



실습 – 모든 하위 디렉터리 검색

- 손으로 돌려보기
 - 동일 행동을 반복
 - 언제까지? → 줄이 빌 때 까지



실습 – 모든 하위 디렉터리 검색

- 반복 행위 구현

```
import os
import queue

dir_queue = queue.Queue()
dir_queue.put("C:/dev/")      # 검색하고자 하는 디렉터리를 줄 앞에 세움

all_dirs = []
while not dir_queue.empty():
    dir_name = dir_queue.get()
    all_dirs.append(dir_name)

    subdir_names = get_subdir(dir_name)
    for each in subdir_names:
        dir_queue.put(each)

print(all_dirs)
```

실습 – 모든 하위 디렉터리 검색

- 특정 디렉터리의 바로 아래 하위 디렉터리 찾기

```
import os
import queue

def get_subdir(path):
    # 검색이 허가되지 않은 디렉토리 접근에 관한 예외처리
    # (이후 예외처리 챕터에서 자세히 다룸)
    try:
        dirfiles = os.listdir(path)
    except PermissionError:
        return

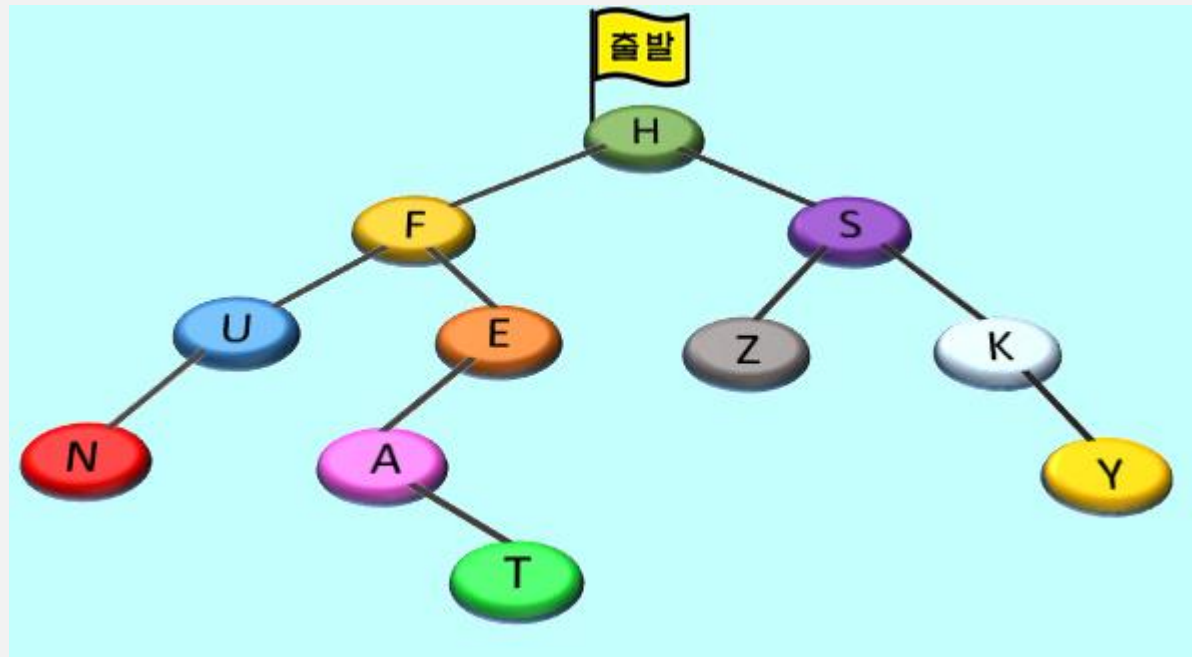
    subdir_list = []
    for each in dirfiles:
        full_name = path + each
        if os.path.isdir(full_name):
            subdir_list.append(full_name + "/")

    return subdir_list
```

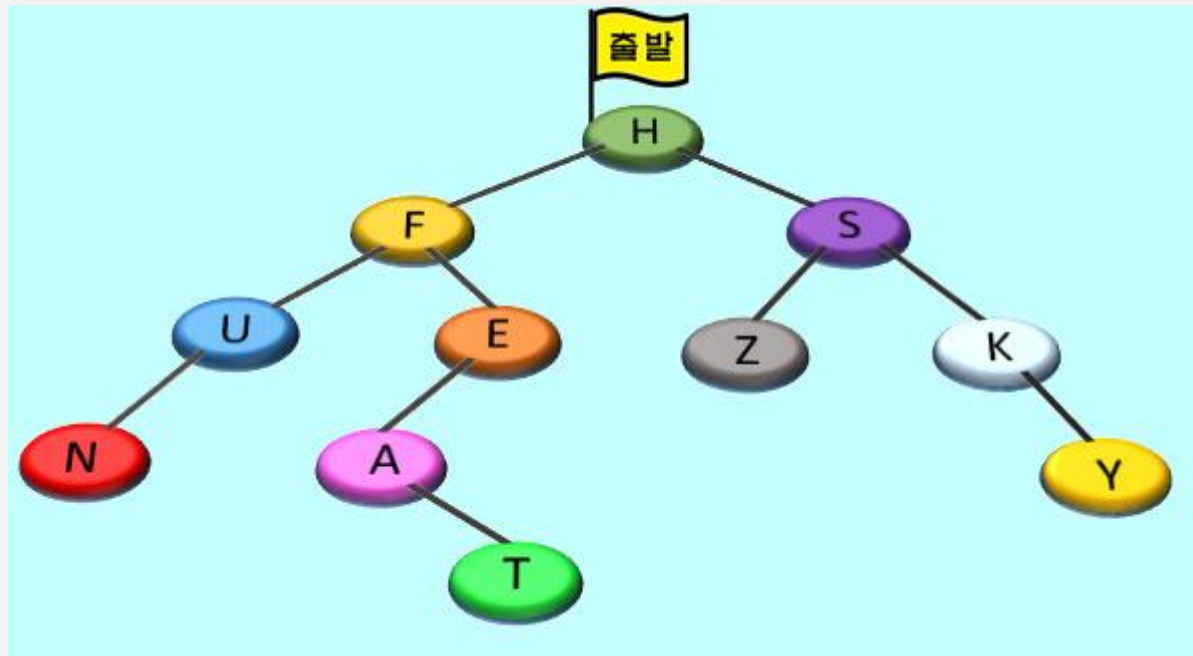

숙제

- 내 컴퓨터에 있는 텍스트 파일 (*.txt) 모두 찾기

- 이진트리에서 트리를 순회하는 3가지 순회 방법
 - 전위순회
 - 중위순회
 - 후위순회



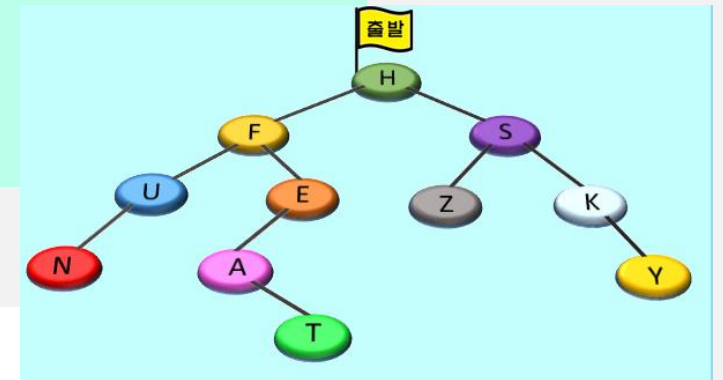
[예제 1] 남태평양에 있는 어느 나라에 11개의 섬이 그림과 같이 다리로 연결되어 있다. 이 나라의 관광청에서는 관광객들이 모든 11개의 섬들의 방문 순서가 다른 3개의 관광코스를 개설하고자 한다.



- 각 코스의 관광은 섬 H에서 시작하며, 관광청에서는 각 관광 코스의 방문 순서를 다음과 같은 규칙 하에 만들었다.

A-코스: 섬에 도착하면 항상 도착한 섬을 먼저 관광하고, 그 다음엔 왼쪽 섬으로 관광을 진행하고 왼쪽 방향의 모든 섬들을 방문한 후에 오른쪽 섬으로 관광을 진행

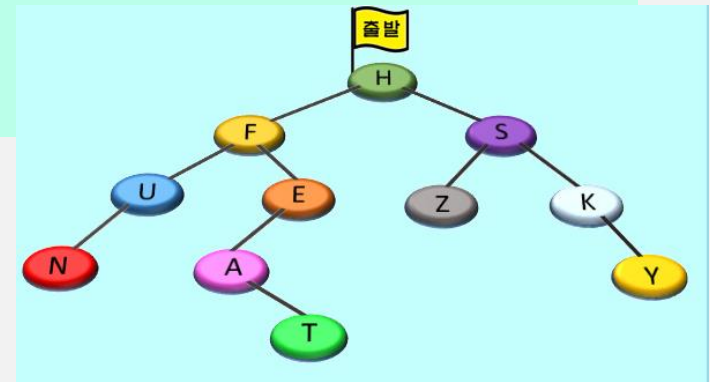
- H, F, U, N, E, A, T, S, Z, K, Y



```
def A_course(n): # A-코스
    if n != None:
        print(n.name, '-> ', end='') # 섬 n 방문
        A_course(n.left) # n의 왼쪽으로 진행
        A_course(n.right) # n의 오른쪽으로 진행
```

B-코스: 섬에 도착하면 도착한 섬의 관광을 미루고, 먼저 왼쪽 섬으로 관광을 진행하고 왼쪽 방향의 모든 섬들을 방문한 후에 돌아와서 섬을 관광한다. 그 다음엔 오른쪽 섬으로 관광을 진행

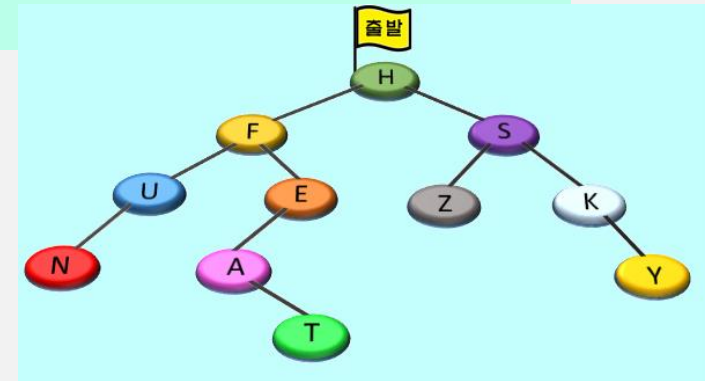
- N, U, F, A, T, E, H, Z, S, K, Y



```
def B_course(n): # B-코스
    if n != None:
        B_course(n.left)      # n의 왼쪽으로 진행
        print(n.name, '-> ', end='') # 섬 n 방문
        B_course(n.right)     # n의 오른쪽으로 진행
```

C-코스: 섬에 도착하면 도착한 섬의 관광을 미루고, 먼저 왼쪽 섬으로 관광을 진행하고 왼쪽 방향의 모든 섬들을 관광한 후에 돌아와서, 오른쪽 섬으로 관광을 진행하고 오른쪽 방향의 모든 섬들을 관광한 후에 돌아와서, 드디어 섬을 관광

- N, U, T, A, E, F, Z, Y, K, S, H



```
def C_course(n): # C-코스
    if n != None:
        C_course(n.left)      # n의 왼쪽으로 진행
        C_course(n.right)     # n의 오른쪽으로 진행
        print(n.name, '-> ', end='') # 섬 n 방문
```

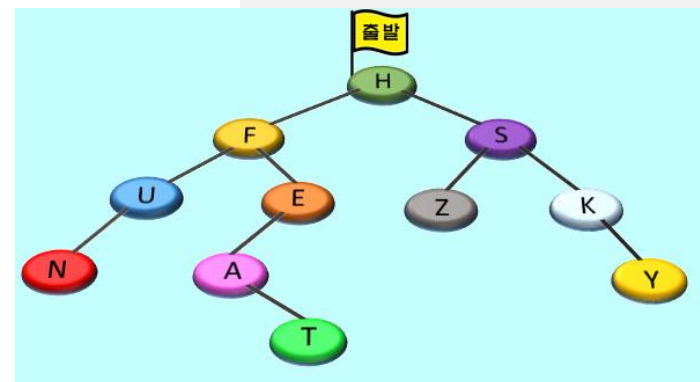
```

01 class Node:
02     def __init__(self, name, left=None, right=None): # 섬 생성자
03         self.name = name
04         self.left = left
05         self.right = right
06
07 def map(): # 지도 만들기
08     n1 = Node('H')
09     n2 = Node('F')
10     n3 = Node('S')
11     n4 = Node('U')
12     n5 = Node('E')
13     n6 = Node('Z')
14     n7 = Node('K')
15     n8 = Node('N')
16     n9 = Node('A')
17     n10 = Node('Y')
18     n11 = Node('T')
19
20     n1.left = n2
21     n1.right = n3
22     n2.left = n4
23     n2.right = n5
24     n3.left = n6
25     n3.right = n7
26     n4.left = n8
27     n5.left = n9
28     n7.right = n10
29     n9.right = n11
30     return n1 # 시작 섬 리턴

```

11개의
섬 만들기

11개의
섬
교량으로
잇기



```

31 def A_course(n): # A-코스
32     if n != None:
33         print(n.name, '-> ', end='') # 섬 n 방문
34         A_course(n.left)           # n의 왼쪽으로 진행
35         A_course(n.right)          # n의 오른쪽으로 진행
36
37 def B_course(n): # B-코스
38     if n != None:
39         B_course(n.left)           # n의 왼쪽으로 진행
40         print(n.name, '-> ', end='') # 섬 n 방문
41         B_course(n.right)          # n의 오른쪽으로 진행
42
43 def C_course(n): # C-코스
44     if n != None:
45         C_course(n.left)           # n의 왼쪽으로 진행
46         C_course(n.right)          # n의 오른쪽으로 진행
47         print(n.name, '-> ', end='') # 섬 n 방문
48
49 start = map() # 시작 섬을 n1으로
50 print('A-코스:\t', end='')
51 A_course(start)
52 print('\nB-코스:\t', end='')
53 B_course(start)
54 print('\nC-코스:\t', end='')
55 C_course(start)

```


island_tour.py

File Edit Format Run Options Window Help

```

1 class Node:
2     def __init__(self, name, left=None, right=None): # 섬 생성자
3         self.name = name
4         self.left = left
5         self.right = right
6
7 def map(): # 지도 만들기
8     n1 = Node('H')
9     n2 = Node('F')
10    n3 = Node('S')
11    n4 = Node('U')
12    n5 = Node('E')
13    n6 = Node('Z')
14    n7 = Node('K')
15    n8 = Node('N')
16    n9 = Node('A')
17    n10 = Node('Y')
18    n11 = Node('T')
19    n1.left = n2
20    n1.right = n3
21    n2.left = n4
22    n2.right = n5
23    n3.left = n6
24    n3.right = n7
25    n4.left = n8
26    n5.left = n9
27    n7.right = n10
28    n9.right = n11
29    return n1 # 시작 섬 리턴
30
31 def A_course(n): # A-코스
32     if n != None:
33         print(n.name, '-> ', end='') # 섬 n 방문
34         A_course(n.left) # n의 왼쪽으로 진행
35         A_course(n.right) # n의 오른쪽으로 진행
36
37 def B_course(n): # B-코스
38     if n != None:
39         B_course(n.left) # n의 왼쪽으로 진행
40         print(n.name, '-> ', end='') # 섬 n 방문
41         B_course(n.right) # n의 오른쪽으로 진행
42
43 def C_course(n): # C-코스
44     if n != None:
45         C_course(n.left) # n의 왼쪽으로 진행
46         C_course(n.right) # n의 오른쪽으로 진행
47         print(n.name, '-> ', end='') # 섬 n 방문
48
49 #main
50 start = map() # 시작 섬을 n1으로
51
52 print('A-코스:Wt', end='')
53 A_course(start)
54
55 print('WnB-코스:Wt', end='')
56 B_course(start)
57
58 print('WnC-코스:Wt', end='')
59 C_course(start)

```

```

A-코스: H -> F -> U -> N -> E -> A -> T -> S -> Z -> K -> Y ->
B-코스: N -> U -> F -> A -> T -> E -> H -> Z -> S -> K -> Y ->
C-코스: N -> U -> T -> A -> E -> F -> Z -> Y -> K -> S -> H ->

```

수행 결과

```
A-코스: H -> F -> U -> N -> E -> A -> T -> S -> Z -> K -> Y ->  
B-코스: N -> U -> F -> A -> T -> E -> H -> Z -> S -> K -> Y ->  
C-코스: N -> U -> T -> A -> E -> F -> Z -> Y -> K -> S -> H ->
```

- 이진트리에서 트리를 순회하는 3가지 순회 방법
 - A-코스 = 전위순회
 - B-코스 = 중위순회
 - C-코스 = 후위순회

감사합니다

Q & A