


파이썬

언어 트렌드

- ◆ <https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019>

Language Ranking: IEEE Spectrum

Rank	Language	Type	Score
1	Python	  	100.0
2	Java	  	96.3
3	C	  	94.4
4	C++	  	87.5
5	R		81.5
6	JavaScript		79.4

파이썬 설치 및 IDE 설치

<https://www.python.org/downloads/>

여러가지 수학 및 과학 패키지들을 기본적으로 포함하는 파이썬 배포판

<https://www.anaconda.com/products/individual>

여러가지 IDE

pycharm, eclipse, visual studio, pydev etc

Pycharm

<https://www.jetbrains.com/ko-kr/pycharm>

Jupyter notebook, python console

Reference document

<https://docs.python.org/ko/3.8/>

jupyter notebook

◆ Python 2버전과 3버전이 같이 설치 되었을때

`python --version`

`python3 --version`

◆ `pip --version`

`pip3 --vrsion`

◆ virtual 환경 만들기

`pip install virtualenv`

`mkdir pro1(프로젝트명)`

`cd pro1`

`python -m venv example(가상환경 이름)`

`cd 환경이름`

`activate.bat`

`deactivate`

◆ 가상환경 목록 확인

`conda info -envs`

◆ anaconda 가상환경 만들기

`conda create -n my_python_env`

◆ 설치 패치지도 같이 설치할때

`conda create -n my_python_env pandas numpy`

`conda create -n my_python_env python=3.4`

◆ 가상환경 삭제

`conda remove -n 가상환경 이름 -all`

◆ 가상환경 활성화

window : activate my_python_env

linux/mac : Source activate my_python_env

◆ 환경 보기 : conda list

◆ 가상환경 비활성화

window: deactivate

linux : source deactivate

◆ 가상환경 삭제

conda env remove -n my_python_env

◆ Jupyter notebook 설치

conda install jupyter notebook

conda install nb_conda

jupyter notebook

- ◆ ctrl+Enter : 현재 셀실행
 - ◆ shift+Enter : 현재 셀 실행 후 아래 셀이동
 - ◆ alter+Enter : 실행 후 아래 셀 추가
 - ◆ Enter: 편집모드
 - ◆ Esc : 편집 모드에서 명령모드로
-
- ◆ dd: 현재 셀 삭제
-
- ◆ a: 현재 셀 위에 셀 추가
 - ◆ b: 현재 셀 아래 셀추가
-
- ◆ x: 선택 셀 잘라내기
 - ◆ c: 선택 셀 복사하기
 - ◆ ctrl +s : 파일저장
-
- ◆ m :마크업 변경
 - ◆ c:코드변경
 - ◆ ctrl+/ :주석처리

Python 특징

- ◆ 배우기 쉽고 간단하다
- ◆ 무료로 사용가능하고 open source이므로 재사용 변경이 가능
- ◆ high-level language : 메모리 관리가 불필요
- ◆ 작성 프로그램은 다양한 플랫폼에서 활용이 가능
- ◆ 풍부한 라이브러리
- ◆ Interpreted language
- ◆ Objected oriented programming
- ◆ Extending and embedding

주석달기

◆ # 한줄 주석

◆ 여러줄 주석

◆ `"""` or `'`

◆ `"""`

안녕하세요
만나서 반갑습니다.
`"""`

◆ # 홍길동 입니다.

◆ 소스 인코딩

◆ `-# coding:latin-1`

◆ `-# -*- coding:utf-8 -*-`

변수 - 정수, 실수, 문자열

숫자형

int float complex

문자형

str

bool형 True, false

시퀀스 형 : list, tuple, range

◆ $0.1 + 0.2 \Rightarrow 0.30000000000000004$

◆ <https://docs.python.org/ko/3/library/decimal.html>

Decimal 모듈 : 실수를 표현하기 위한 float 자료형 보다 정확히 사용함

◆ `from decimal import getcontext, Decimal`

◆ `Decimal(3);`

◆ `a=Decimal('1.1');`

◆ `a*2;`

출력 포맷문자 리터럴 사용

- ◆ f-string

- ◆ `name='hong gil dong';`
`age=10;`

```
print(f' my name is {name}');  
print(f" i'm {age} years old ");
```

- ◆ `say='\"안녕하세요 만나서 반갑습니다.\"';`
`say2='W'helloW';`

```
print(say);  
print(say2);
```

출력 포맷문자 리터럴 사용

◆ `name='aa';`
`age=10;`
`print('%s 님은 나이가 %d' %(name, age));`

◆ 문자열 format

◆ `%s` : 문자열 `%c`: 문자 1개

◆ `%d` : 정수 `%f` :부동소수

◆ `%o` : 8진수 `%x` : 16진수

◆ `%%` : literal %

◆ `p=20;`
`print('%d %%' %(p));`

출력 포맷문자 리터럴 사용

- ◆ 문자열의 `str.format()` 메서드 사용
- ◆ `for index in range(1,11):`
 `print("{0:4d} {1:10d} {2:4d}".format(index, index*2, index*index));`
- ◆ `print('{1} {0} and {other}'.format('aaa','bbb', other='hello'));`
- ◆ `for i in range(0,10):`
 `print(i , end='/');`

연산자

◆ 산술연산자

- +(더하기) -(빼기) * (곱하기) / (나누기)
- **(제곱승), %(나머지)
- x//y : 몫
- abs, divmod, pow

ex 2**3, 2%3, abs(-3), pow(4,2) divmod(5,3);

◆ 대입연산자

- += -= *= /= //= %= **=

◆ 논리연산자

- and, or, not

◆ 비교연산자

- <(작다) <=(작거나 같다) >(크다) >=(크거나 같다) ==(같다) !=(같지않다)
- 객체 비교 : is not is

<https://docs.python.org/ko/3/library/stdtypes.html#truth-value-testing>

연산자

◆ 비트연산자

- & | ~ ^
- << >>

◆ membership 연산자

- in not in

help(함수명)

int(값)

float(값)

str(값)

조건문

if 조건식 :
 명령문

if 조건식:
 명령문 1;
else:
 명령문 2;

if 조건1:
 명령문1;
elif 조건2:
 명령문2;
elif 조건3:
 명령문3;
else:
 명령문4;

조건문

◆ if

```
x=int(input('input su'));
```

```
if x < 0:
    print('음수입니다.');
```

```
else:
    print('양수입니다');
```

```
name=input('input name');
jumsu=int(input('input jumsu'));
```

```
if(jumsu>=90):
    result="수";
elif(jumsu>=80):
    result="우";
elif(jumsu>=70):
    result="미";
elif(jumsu>=60):
    result="양";
else:
    result="가";
```

```
print("jumsu:{0}".format(jumsu));
print("result:{0}".format(result));
```

반복문

- ◆ while : 조건이 참일 동안 계속적으로 반복 수행

while 조건식:
구문

예
i = 10
while i > 0:
 print(i)
 i = i - 1;

무한루프

i = 10

while True:

 print(i, end=' ')

 n = n - 1

print('done')

⇒ 해결 break

i = 10

while True:

 print(i, end=' ')

 i = i - 1

 if i == 5:

 break

print('done')

반복문

◆ for문

for 아이템 in 시퀀스형 객체 :
 명령문

for문에서 사용할 수 있는 자료
=> 문자열, 리스트, 튜플, 사전
=> 이터레이터, 제너레이터 객체

```
name="안녕하세요 python 님 만나서 반갑습니다"
for index in name:
    print(index);
```

```
friends=['hong','kim','lee','park']
for index in friends:
    print(index);
print('end');
```

break, continue

- ◆ **continue** : 반복문 내부 블록을 수행하지 않고 블록 시작점으로 이동

```
i=10
```

```
while i>0:
```

```
    i=i-1
```

```
    if i%3==0:
```

```
        continue
```

```
    print(i)
```

- ◆ **break**를 만나면 반복문 블록을 빠져나감

```
i=10
```

```
while i>0:
```

```
    i=i-1
```

```
    if i==3:
```

```
        break
```

```
    print(i)
```

함수

- ◆ 여러 개의 문장(statement)을 하나로 묶어주는 것
- ◆ 이미 정의되어 있는 함수(built-in), 필요한 함수를 정의 가능
- ◆ 함수를 부르는 것을 호출이라고 한다.
- ◆ 함수를 선언할때 def로 시작하고 ;(콜론)으로 끝낸다.

함수 선언

```
def 함수명():
```

```
    명령문1
```

```
    명령문2
```

```
def 함수명(매개변수1, 매개변수2, .... 매개변수n):
```

```
    명령문1
```

```
    명령문2
```

```
    명령문3
```

```
    ....
```

```
def 함수명(*매개변수): // 매개변수가 몇 개인지 모를때 -가변인수
```

```
    명령문1
```

```
    ...
```

함수

- ◆ 파라미터를 몇 개 받을지 모르는 경우
튜플형식으로 전달

```
def test(*args):  
    print(args)
```

```
=> test('a','b','c')
```

```
=> a=({'name':'hong','age':10})
```

```
    test(a)
```

```
    test(*a) # 튜플형식으로 이루어짐
```

- ◆ 딕셔너리 형태로 전달될때

```
def test(**kwargs):  
    print(kwargs)
```

```
=> test(name='a',age=20)
```

```
=> a=({'name':'hong','age':10})
```

```
    test(**a)
```

함수

- ◆ **return** : 함수를 종료시키고 호출한 곳으로 돌아가게 함
어떤 객체로도 돌려줄 수 있음, 여러 개 값을 튜플로 전달 할 수 있음

```
def 함수명 ()  
    명령문1  
    명령문2  
    return 반환값
```

```
def 함수명(매개변수1, 매개변수2... 매개변수 n):  
    명령문1  
    명령문2  
    ....  
    return 반환값
```

- ◆ **pass** : 아무것도 하지 않는 함수, 모듈, 클래스만들때
def sampe():
 pass

텍스트 시퀀스 형 -str

◆ 문자열은 ' ' or " " or ''' ''' or """" """" 로 표현함

◆ `a='kim';`
`b='hana';`
`print(a+b);`
`print(type(a+b)); #str`

◆ `a='kim';`
`print(a*2);`

`type(20);`
`dir(10.2);`
`dir('abc');`

예제

`a='abc';`
`dir(a);`
`help(str);`

텍스트 시퀀스 형 -str

- ◆ 모든 문자열은 기본적으로 유니코드
- ◆ 다른 인코딩으로 되어 있는 경우 문자열을 bytes로 표현

```
a='가'.encode('utf-8')  
print(a)  
print(type(a))
```

- ◆ 문자열 -

```
fruit='apple'  
letter=fruit[1] //p
```

```
len(fruit ) //길이
```

```
a='hello python';  
print("{0}....{1}".format(a[:2],a[6:]))  
a[0]='t' // 'str' object does not support item assignment  
print(a)
```

텍스트 시퀀스 형 -str

- ◆ 문자열 함수 `help(str.메서드명)`
 - `str.capitalize()` : 첫번째가 대문자 나머지가 소문자
 - `S.count(sub[,start[, end]])` – int
 - `startswith(...)` method of builtins.str instance
 - `S.startswith(prefix[, start[, end]])` -> bool
 - ex `help(a.startswith);`
- ◆ `a='abcabc';`
 - `a.startswith('a',2);`
 - `a.startswith('a',2,4);`
- ◆ **replace : 문자열 바꾸기**
 - `replace(old, new, count=-1, /)` count Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.
- ◆ **split : 문자열 나누기**
 - `split(sep=None, maxsplit=-1)`
 - method of builtins.str instance Return a list of the words in the string, using sep as the delimiter string.

<https://docs.python.org/ko/3.9/library/stdtypes.html#text-sequence-type-str>

텍스트 시퀀스 형 -str

◆ **count**: 문자갯수 세기

- S.count(sub[,start[, end]]) – int

◆ **upper /lower/capitalize** : 대문자/소문자/첫문자 대문자 나머지 소문자

- upper() method of builtins.str instance

Return a copy of the string converted to uppercase.

◆ **find** : 문자 위치 찾지못하면 -1

- S.find(sub[, start[, end]]) -> int

◆ **index** : 문자위치 찾지못하면 오류발생

- S.index(sub[, start[, end]]) -> int
- Raises ValueError when the substring is not found.

◆ **lstrip/rstrip/strip** : 왼쪽 공백지우기 /오른쪽 공백지우기/양쪽 공백지우기

- strip(chars=None, /)
- method of builtins.str instance Return a copy of the string with leading and trailing whitespace removed.

◆ **join**:

- The string whose method is called is inserted in between each given string. The result is returned as a new string.
- Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

리스트(list)

- ◆ 공통 관계를 가진 값들의 모임
- ◆ 값들의 나열
- ◆ [10,20,30,40]
- ◆ ['a1','a2','a3',[10,20,30],'a4']

- ◆ type(list)

- ◆ slice

```
a1=[10, 20, 30, 40, 50];  
a2=['a', 'b', 'c', 'd'];  
type(a1); # 결과 class 'list'  
print(a1);  
a1[1:3]; # 결과 [20,30]  
#a1[start: end] start: include end : exclude  
a1[:2]; # 결과 [10,20];  
a1[1:]; #결과 [20,30,40];
```

연산자

◆ + :list추가

```
a=[10,20,30];
```

```
b=[1,2,3];
```

```
c=a+b
```

```
print(c)
```

```
a=[10,20,30]
```

```
b=a*2
```

```
print(b) //10,20,30,10,20,30
```

◆ slice

◆ List method

리스트

◆ 연산

- `x in s` : s항목중 하나가 x와 같으면 `true` (`x not in s`)
- `s+t` : s와 t 이어 붙이기
- `s*n` : s를 그 자신에 n번 더하는 것
- `len(s)` : s의 길이
- `min(s)` : s의 가장 작은 항목
- `s.index(x[, i[, j]])` : s에서 인덱스 i또는 그이후부터 인덱스 j 전까지 등장하는 첫번째 x의 위치
- `s.count(x)` : s에 등장하는 x의 총수

◆ 메서드

- `append(self, object, /)` 리스트 맨 마지막에 object를 추가
- `clear(self, /)` list의 모든 요소를 삭제

<https://docs.python.org/ko/3.9/library/stdtypes.html#sequence-types-list-tuple-range>

리스트

- `count(self, value, /)` : 리스트 안에 `value`가 몇 개 있는지 개수를 return
- `extend(self, iterable, /)` `iterable`에 리스트를 넣어서 원래 리스트를 확장
`iterable`객체-반복가능한 객체 ,`list`, `dic`, `set`, `str`, `bytes`, `tuple`, `range`
- `index(self, value, start=0, stop=9223372036854775807, /)`
리스트의 처음으로 만나는 `value`의 `index`값 반환
`value`가 없으면 `ValueError` 발생
- `insert(self, index, object, /)` `index`에 `object` 추가 기존에 있는 요소는 뒤로 밀림
- `pop(self, index=-1, /)` : 리스트의 맨 마지막 요소를 돌려주고 그 요소는 삭제
`index`위치를 지정시 `index` 요소를 돌려주고 그 요소는 `list`에서 삭제
리스트가 비어있거나 `index`가 범위를 벗어나면 `IndexError` 발생
- `remove(self, value, /)` `list`에서 첫번째 `value`값을 삭제 , `value`값이 없으면 `ValueError` 발생
`del 리스트명[index]` 과 동일
- `reverse(self, /)` : 리스트를 거꾸로 뒤집는다.
- `sort(self, /, *, key=None, reverse=False)` : 정렬 `reverse=True`-내림차순
`key` 매개 변수의 값은 단일 인자를 취하고 정렬 목적으로 사용할 키를 반환하는 함수

<https://docs.python.org/ko/3/howto/sorting.html>

튜플

- ◆ 리스트와 유사, 읽기 전용
- ◆ 값 변경이 안됨
- ◆ 괄호 () 로 표현
- ◆ t1=(1,2 ...)
- ◆ 속도가 빠름

```
a,b=1,2
print(a,b) //1 2
a,b=b,a
print(a,b) //2 1
```

```
t1=(10,20,(10,20,30))
print(type(t1)) # <class 'tuple'>
del t1[0] #TypeError: 'tuple' object doesn't support item deletion
t1[0]='aa' #TypeError: 'tuple' object does not support item assignment
```

메서드 -count, index

딕셔너리

◆ key, value로 이루어진 자료형

◆ key를 통해 value값을 얻음

◆ dic={key1:value1, key2:value2, key3: value3...}

◆ 사용예제

```
temprice={"a1":1000, "a2":13000, "a3":2000}
```

```
print(type(itemprice)) # <class 'dict'>
```

```
print(itemprice["a1"]) # 1000
```

◆ 딕셔너리 수정- 기존의 키값에 새로운 값 작성

```
itemprice["a1"]=3000
```

```
print(itemprice) # {'a1': 3000, 'a2': 13000, 'a3': 2000}
```

◆ 딕셔너리의 키, 값을 추가 -새로운 키와 그에 해당하는 값을 작성

```
itemprice["t1"]=200
```

```
print(itemprice) // {'a1': 3000, 'a2': 13000, 'a3': 2000, 't1': 200}
```

◆ 딕셔너리 키, 값 삭제

```
del itemprice["t1"] //t1 키에 해당하는 키, 값을 삭제
```

딕셔너리

◆ 메서드

- `clear` : 딕셔너리 객체의 모든 요소를 삭제
- `copy` : 기존 딕셔너리객체를 `copy` 새로운 객체 생성
- `get(key, default=None, /)` 딕셔너리에서 `key`에 해당하는 `value`를 `return`
해당 `key`가 없으면 `None`
- `items()` : `key`와 `value`의 쌍을 튜플로 묶은 값을 `dict_items`객체로 리턴
=> `dict_items([('a1', 3000), ('a2', 13000), ('a3', 2000)])`
- `keys()`
python 3.0이후 `list`가 아닌 `dict_keys(['a1', 'a2', 'a3'])`로 `key`값을 반환한다.
`for`문 사용은 가능하지만 `insert`, `pop`, `remove`등의 함수는 사용할 수 없다.
=>리스트 변환을 위해서는 `list(a.keys())`
- `pop(k[,d])` : `key`에 해당하는 `value`를 리턴하고 딕셔너리에서는 삭제
딕셔너리에 해당 `key`가 없으면 `d`작성시 `d`리턴, `d`작성이 안되어 있으면 `KeyError`발생
- `popitem` : 딕셔너리에서 끝의 요소의 `key,value`를 리턴하고 딕셔너리에서 삭제
- `values()` 딕셔너리에서 값만 얻고자 할때 `dict_values`객체로 리턴
=> `dict_values([3000, 13000, 2000])`

문제

Q1. 단어의 빈도수를 계산하시오

a="딸기 포도 복숭아 딸기 사과 배 포도 딸기 바나나";

처리결과

{'딸기': 3, '포도': 2, '복숭아': 1, '사과': 1, '배': 1, '바나나': 1}

Q2. fruit_price={"사과":3000, "딸기":1000, "바나나":2500,"복숭아":40000}

자료에서 high_price와 low_price란 이름의 list를 만들고

가격이 3000이상인 과일은 high_price에 그외 과일은 low_price에 넣어서
출력하시오.

range, Enumerate, zip

- ◆ range : range(시작숫자, 종료숫자, step)
- ◆ enumerate : 인덱스 번호와 컬렉션 원소를 tuple 형태로 반환

```
a=['hong','kim','lee','park']
```

```
b=enumerate(a)
```

```
print(type(b))
```

처리결과

0 hong

1 kim

2 lee

3 park

```
for i, j in enumerate(a):
```

```
    print(i , j)
```

- ◆ zip : 동일한 개수로 이루어진 자료형을 묶어주는 역할
zip(iterable)

```
lst1=["hong","kim","lee","park"]
```

```
lst2=[20,50,40,12]
```

```
for i, j in zip(lst1, lst2):
```

```
    print(i, j)
```

packing unpacking

- ◆ `a,b=10,20 #a:10, b:20`
- ◆ `e=(20,30)`
 - `t1,t2=e #t1:20, t2:30`
- ◆ `a, b=10, 20`
 - `b, a=a, b # a:20, b=10`

클래스

◆ class Dog:
 pass

◆ 인스턴스 객체 생성
d1=Dog()
print(d1) #<__main__.Dog object at 0x000001B09C9ACDC0>

◆ class Person: #클래스
 name="hong gil dong" #멤버변수
 def prt(self): #메서드
 print('my name is {0}'.format(name))

◆ 인스턴스 객체생성
p1=Person()
P2=Person()
print(p1 is p2) #False

클래스

◆ 메서드

- 파이썬 메서드의 첫 번째 매개변수 이름은 관례적으로 `self`를 사용한다.
- 객체를 호출할 때 호출한 객체 자신이 전달되기 때문에 `self`를 사용

```
class Person:
```

```
    def setData(self, name, age):
```

```
        self.name=name;
```

```
        self.age=age;
```

```
    def getData(self):
```

```
        print("self : {0}  name:{1}  age:{2}".format(self, self.name, self.age))
```

```
p1=Person()
```

```
p1.setData('hong',20);
```

```
p1.getData()
```

```
isinstance(p1, Person) # True  인스턴스 객체가 해당 클래스로부터 생성되었는 판별
```

클래스

◆ 생성자(Constructor)

- 객체 생성시 초기화 작업을 수행
- 인스턴스 객체가 생성시 자동으로 호출
- `__init__()`

◆ 소멸자

- 소멸시 종료작업 수행
- 인스턴스 객체의 참조 카운터가 0일때 호출
- `__del__()`

◆ 클래스 변수

- 클래스 안에 변수 선언
 `class Dog:`
 `kind="시츄"`
- 사용방법
 클래스이름.변수명

클래스

◆ 문자열화 함수 : `__str__`

- 인스턴스 자체를 출력할 때의 형식을 지정해 주는 함수

```
class Man:
```

```
    def __init__(self, name, age):
```

```
        self.name=name
```

```
        self.age=age
```

```
    def __str__(self):
```

```
        return self.name+","+str(self.age)
```

```
m1=Man("hong",20)
```

```
print(m1)
```

클래스

◆ 정적메서드

@staticmethod

인스턴스 생성하지 않고 class이름.메서드()

작성

```
class Test:
```

```
    num=100
```

```
    @staticmethod
```

```
    def add(x, y):
```

```
        return x+y+Test.num #num접근시 오류
```

```
print(Test.num)
```

```
print(Test.add(1,2))
```

클래스

◆ 클래스 메서드

@classmethod

인스턴스 생성하지 않고 class이름.메서드()

작성

```
class Test:
```

```
    num=100
```

```
    @classmethod
```

```
    def add(cls, x, y): #cls는 클래스를 가르킴
```

```
        return x+y+cls.num
```

```
    @classmethod
```

```
    def prt(cls):
```

```
        print(cls)
```

```
print(Test.num)
```

```
print(Test.add(1,2))
```

```
Test.prt() #<class '__main__.Test'>
```

클래스

◆ 상속

class 자식클래스명(부모클래스):
 나머지 정의...

class 자식클래스(부모클래스1, 부모클래스2): # 2개 이상의 클래스상속
 나머지 정의...

상속여부 확인 : `issubclass(자식클래스, 부모클래스)`

```
class Man:
    def t1(self):
        print("Man class ")
```

```
class Student(Man):
    def t2(self):
        print("Student")
```

```
s1=Student()
```

```
s1.t1()
```

```
s1.t2()
```

클래스

◆ super : 부모 클래스 내용 호출시

```
class A:
    def __init__(self):
        print('a')

class B(A):
    def __init__(self):
        super().__init__()
        print('b')

b=B()
```

클래스

◆ 메서드 오버라이딩 :부모 클래스에서 정의된 메서드 자식클래스에서 다시 정의

```
class A:
```

```
    def prt(self):
```

```
        print("A class prt")
```

```
class B(A):
```

```
    def prt(self):
```

```
        super().prt()
```

```
        print("B class prt")
```

```
a=A()
```

```
a.prt()
```

```
b=B()
```

```
b.prt()
```

추상클래스

◆ from abc import *
class Animal(metaclass=ABCMeta):
 @abstractmethod
 def sound(self):
 pass

class Dog(Animal):
 def walk(self):
 print('걸습니다')

d = Dog()

def sound(self):
 print('walwal')

Dog 클래스에 sound 메서드 오버라이딩
처리

◆ d = Dog()

에러

TypeError: Can't instantiate abstract class Dog with abstract methods sound
오버라이딩 필요

클래스

◆ 연산자 중복정의

class Complex:

def __init__(self,real=0,image=0):

self.__real = real

self.__image = image

def __add__(self,other):

real = self.__real+other.__real

image = self.__image+other.__image

return Complex(real,image)

def __sub__(self, other):

real = self.__real-other.__real

image = self.__image-other.__image

return Complex(real,image)

def __str__(self):

if(self.__image>=0):

return str(self.__real)+" "+str(self.__image)+"i"

else:

return str(self.__real)+str(self.__image)+"i"

n1 = Complex(2,3)

n2 = Complex(3,4)

print("n1:",n1)

print("n2:",n2)

print("n1+n2:",n1+n2)

print("n1-n2:",n1-n2)

모듈

◆ 재사용할 목적으로 작성된 소스 파일

◆ 함수, 클래스 등으로 구성

◆ 외부 모듈 불러오기

모듈.이름 형식으로 모듈안에 데이터나 함수를 사용

from 모듈 import 어트리뷰트

import 모듈 as 별칭

- import random #random.py
- import pandas # pandas.메서드로 사용
- import pandas as pd #pd.메서드로 사용가능
- from pandas import DataFrame # 모듈안 특정 함수만 사용하고 싶을때
- from pandas import *

◆ **from pandas import DataFrame**

```
d=DataFrame([["hong","kim","lee"],[20,15,26],["서울","경기","강원"]],columns=["name","age","addr"])
```

```
print(d)
```

◆ 외부 모듈 설치하기 `pip install wordcloud`

https://amueller.github.io/word_cloud/

모듈

◆ module1.py

```
def sayHi():  
    print("hi method")
```

```
def sayHello():  
    print("hello method")
```

◆ moduletest.py

import module1 #module1.py에서 확장자를 뺀 나머지가 모듈이름

```
module1.sayHello()  
module1.sayHi()
```

◆ import 모듈명

dir(모듈) : 모듈에 어떤 함수가 있는지 확인 가능

◆ 모듈 경로 밖의 모듈은 임포트 할 수 없음

탐색경로

1. 프로그램 실행된 디렉토리
2. `pythonpath` 환경 변수에 등록된 위치
3. 표준라이브러리 디렉토리 순으로 검색 순으로 검색

모듈_main추가

◆ module1.py

```
def sayHi():  
    print("hi method")
```

```
def sayHello():  
    print("hello method")
```

```
if __name__ == "__main__":  
    sayHi()  
    sayHello()
```

◆ moduletest1.py

```
import module1
```

◆ if __name__ == "__main__": 선언이 없을 경우 import 시에도 함수가 그대로 실행됨

◆ 따라서 if __name__ == "__main__": 선언을 함으로 import시에는 함수 실행을 하지 않고 module1.py를 실행할때만 함수 실행이 될 수 있도록 해야 함

패키지

- ◆ 파이썬의 모듈 이름공간을 구조화 하도록 하는 방법
새로운 디렉토리에 python파일을 작성

실습

1. pythonProject디렉토리에 view.py파일을 생성

2. view.py

```
def prt():  
    print("prt ");
```

```
def draw():  
    print("draw function")
```

3. 현재 project 공간에서 PYTHONPATH 환경변수 등록
set PYTHONPATH=경로₩ pythonProject

4. 현재 python파일에서 import 후 사용

```
import view  
view.prt()  
view.draw()
```

예외처리

- ◆ 프로그램 제어 흐름을 조정하기 위해 사용하는 이벤트
- ◆ `a=[10,20,30]`
`print(a[3])`
`IndexError: list index out of range`
- ◆ `while True :`
`su1=int(input('input su1'))`
`su2=int(input('input su2'))`
`print(su1/su2)`
`=> 프로그램이 중간에 멈춤`

<https://docs.python.org/ko/3/library/exceptions.html#base-classes>

예외처리

◆ 내장 예외는 exceptions 모듈에 미리 정의

◆ 주요 내장 예외

- **Exception** : 모든 내장 예외의 기본 클래스
- **ArithmeticError**: 수치 연산 예외 클래스
- **LookupError**: 매핑 또는 시퀀스에 사용된 키나 인덱스가 잘못되었을때
- **BufferError** : 버퍼 관련 연산에 관한 예외 클래스

내장 예외의 클래스 계층 구조는 다음과 같습니다:

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
        |   +-- FloatingPointError
        |   +-- OverflowError
        |   +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError
    +-- ImportError
        |   +-- ModuleNotFoundError
    +-- LookupError
        |   +-- IndexError
        |   +-- KeyError
    +-- MemoryError
    +-- NameError
        |   +-- UnboundLocalError
```

<https://docs.python.org/ko/3.8/library/exceptions.html#base-classes>

예외처리

◆ try:
 발생 오류가 일어날 가능성이 있는 문장
except:
 오류발생시 처리할 문장

◆ try:
 발생 오류가 일어날 가능성 있는 문장
except 발생 오류 as 오류 메시지 변수:
 오류 발생시 처리할 문장

구현문장

```
try:  
    4 / 0  
except ZeroDivisionError as e:  
    print(e)
```

예외처리

◆ try:

문장

except:

오류발생시 처리할 문장

finally:

오류와 관계없이 처리해야 할 문장

a, b=10,2

try:

print(a/b)

except Exception as e:

print(e)

finally:

print('end')

처리결과

5.0

end

a, b=10,0

try:

print(a/b)

except Exception as e:

print(e)

finally:

print('end')

처리결과

division by zero

end

예외처리

- ◆ 여러 개 오류처리

- ◆ try:

- 문장

- else:

- 예외가 없을때 사용할 구문

- except 발생 오류1:

- 발생오류1에 대한 처리문

- except 발생오류2:

- 발생오류2에 대한 처리문

- except: #발생오류1,2에 해당하지 않는 오류들

- 나머지 오류에 관한 처리문

- finally:

- 오류와 관계없이 처리해야 할 문장

예외처리

◆ try:

```
a = [1,2]
```

```
print(a[3]) 4/0
```

```
except ZeroDivisionError as e:
```

```
    print(e)
```

```
except IndexError as e:
```

```
    print(e)
```

◆ try:

```
a = [1,2]
```

```
print(a[3])
```

```
4/0
```

```
except (ZeroDivisionError, IndexError) as e:
```

```
    print(e)
```

예외처리

해결!

```
while True :  
    try:  
        su1=int(input('input su1'))  
        su2=int(input('input su2'))  
        print(su1/su2)  
    except ValueError as e1:  
        print('정수입력 {0}'.format(e1))  
    except ZeroDivisionError as e2:  
        print('0으로 나눔 {0}'.format(e2))  
    except:  
        print('다른 이유의 에러')  
    finally:  
        a=input('프로그램 종료를 원하면 .')  
        if a=='.':  
            break
```

예외처리

◆ raise : 명시적으로 예외 발생

```
raise[Exception]
```

```
raise[Exception(data)]
```

◆ try:

```
a=int(input('3이상 입력하세요'))
```

```
if a<3:
```

```
    raise ZeroDivisionError('3이상 입력되어야 합니다')
```

```
    print("입력한 값에 10을 더한 값{0}".format((a+10)))
```

```
except ZeroDivisionError as e1:
```

```
    print(e1)
```

```
except Exception as e2:
```

```
    print('입력오류{0}'.format((e2)))
```

예외처리

- ◆ 개발자가 직접 예외를 정의하여 사용
- ◆ Exception 클래스 또는 그 하위 클래스를 상속받아서 구현
- ◆ `class UserException(Exception):`

```
    def __init__(self, value):  
        self.value=value
```

- ◆ `class UserException(Exception):`
 `def __init__(self, value):`
 `self.value=value`

```
try:  
    a=int(input('3이상 입력하세요'))  
    if a<3:  
        raise UserException('3이상이 입력되어야 합니다')  
    print("입력한 값에 10을 더한 값{0}".format((a+10)))  
except UserException as e1:  
    print(e1)  
except Exception as e2:  
    print('입력오류{0}'.format((e2)))
```

예외처리

- ◆ `assert <condition>, <error message>`
- ◆ Assert 뒤의 조건식이 True가 아니면 `AssertionError`를 발생시킴
- ◆ `def test(t):`
 - `assert type(t) is int, '정수 아닌 값 포함'`
 - `print(t)`

```
lst=[10,20,30,'e']  
#lst=[10,20,30,40]
```

```
for i in lst:  
    test(i)
```

```
10  
20  
30
```

```
Traceback (most recent call last):  
  assert type(t) is int, '정수 아닌 값 포함'  
AssertionError: 정수 아닌 값 포함
```

리스트 내장

◆ [표현식 for 아이템 in 시퀀스 객체 (if 조건식)]

◆ lst=[6,2,1,2,4]

```
result=[i+10 for i in lst]  
print(result)
```

◆ lst=['hong','kim','lee','park']

```
t2=[i for i in lst if len(i)>=4]  
print(t2)
```

◆ lst=['hong','kim','lee','park']

```
t2=[len(i) for i in lst]  
print(t2)
```

람다

- ◆ lambda 함수
- ◆ 이름이 없는 1줄짜리 함수
- ◆ Lamda 인수: 구문
- ◆ 사용이유
 - 함수를 매개값으로 넘길때
 - 간단한 함수가 필요할때
 - 프로그램 가독이 쉽다

```
g=lambda x,y : x+y  
print(g(2,3))
```

```
t=(lambda x:x*x)(5)  
print(t)
```

조건식

```
t2=(lambda x,y: x if x>y else y)(7,3)  
print(t2)
```


map

◆ map(function, iterable)

iterable의 모든 item에 function을 적용한 iterator를 return 해줌

```
lst={'name':['hong','kim','lee','park'], 'age':[10,22,13,20]}  
r=map(lambda i:i+10, lst['age'])  
for index in r:  
    print(index)
```

문제

jumsu=[50,40,100,20,70,90,30] 가 있을때 map을 이용하여 60이상 'pass'
60점 미만을 'fail'로 처리한 값을 jumsu와 같이 출력하시오

출력결과

(50, 'fail')(40, 'fail')(100, 'pass')(20, 'fail')(70, 'pass')(90, 'pass')(30, 'fail')

<https://docs.python.org/3/library/functions.html>

filter

- ◆ `Filter(function, iterable)`
- ◆ Iterable타입인 자료형 요소를 함수를 적용하여 맞는 자료만 걸러서 return 한다.
- ◆ 함수 리턴은 True, False이어야 한다.

```
def passfail(x):  
    return x>60
```

```
t=list(filter(passfail,[100,60,30,20,76]))  
print(t)
```

<https://docs.python.org/ko/3.8/library/functions.html#filter>

Iterator

- ◆ <https://docs.python.org/3/tutorial/classes.html#iterators>
- ◆ 내부 반복문을 관리해 주는 객체
- ◆ `__next__()`를 이용해 순회 가능한 객체의 요소를 하나씩 접근할 수 있음

- ◆

```
lst=[10,20,30,40]
print(type(lst))    #<class 'list'>
print(type(iter(lst))) #<class 'list_iterator'>
```

- ◆

```
lst=iter(['a','b','c','d'])
while True:
    try:
        print(next(lst)) # __iter__() 메서드 호출
    except:
        break
```

gernerator

- ◆ Iterator 를 생성해주는 함수
- ◆ Iterator는 클래스에 `iter`, `next`, `__getitem__` 메서드를 구현해야 하지만 제너레이터는 함수 안에 `yield` 키워드만 사용하면 된다.

- ◆ 영문자 소문자 자료를 역순으로 대문자 출력하는 프로그램

```
def upperReverse(data):  
    for index in range(len(data)-1, -1, -1):  
        if ord(data[index])>=97 and ord(data[index])<=122:  
            yield chr(ord(data[index])-32)  
        else:  
            yield data[index]
```

```
for index in upperReverse('ABCD'):  
    print(index, end='')
```

```
g=upperReverse("abc")  
while True:  
    try:  
        print(next(g))  
    except:  
        break
```

<https://docs.python.org/ko/3/reference/expressions.html#yieldexpr>

입출력-print

◆ `print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`

◆ `*objects`는 튜플형식으로 가변화된 인수

◆ `sep`으로 구분지으며 `print`후 `end`내용을 붙임

```
print("hello","hong","kim",sep=";", end="...")
```

=> 출력결과

```
hello;hong;kim...
```

입출력-formatting

- ◆ {}안의 인덱스를 지정하여 출력
- ◆ `print("{1}, {0}, {2}".format("aa","bb","cc"))`
=>bb, aa, cc
- ◆ `print("{name} {age}".format(name="hong",age=20))`
=>hong 20
- ◆ `product={"code":"p1","price":2000,"name":"볼펜"}`
`print("code:{0[code]}, price:{0[price]}, name:{0[name]}".format(product))`
=>code:p1, price:2000, name:볼펜
- ◆ `product={"code":"p1","price":2000,"name":"볼펜"}`
`print("code:{product[code]}, price:{product[price]}, name:{product[name]}".format(product=product))`
=>code:p1, price:2000, name:볼펜
- ◆ `product={"code":"p1","price":2000,"name":"볼펜"}`
`print("code:{code} price:{price} name:{name}".format(**product))`
=> code:p1 price:2000 name:볼펜

<https://docs.python.org/ko/3/tutorial/inputoutput.html>

파일 입출력

◆ 파일 열기

객체=open(file, mode)

◆ 파일 닫기

객체.close

◆ mode

- r: 읽기모드(read) -default
- a : 쓰기+이어쓰기 모드(append)
- b : 바이너리모드
- w : 쓰기모드
- + : 읽기+쓰기모드
- t : 텍스트 모드 -default

<https://docs.python.org/3.3/tutorial/inputoutput.html#reading-and-writing-files>

파일 입출력

◆ 파일 읽기

readline : reads a single line from the file; a newline character (**\n**)

readlines:

read([size])

◆ 파일쓰기

f.write(string) : writes the contents of *string* to the file, returning the number of characters written.

◆ with.. open ...as

파일을 open했을때 close를 해야 하지만

with..open는 with를 빠져나가는 순간 자동 close됨

<https://docs.python.org/3.3/tutorial/inputoutput.html#reading-and-writing-files>

파일 입출력

◆ 파일 입출력 - 텍스트

◆ `f=open('./data/paper.txt','r',encoding='utf-8')` #윈도우 표준 인코딩 cp949

`while True:`

`line=f.readline()`

`if not line: break`

`print(line)`

`f.close()`

◆ `f=open('./data/paper.txt','r',encoding='utf-8')` #윈도우 표준 인코딩 cp949

`lines = f.readlines()`

`for line in lines:`

`print(line)`

`f.close()`

파일 입출력

◆ line_count=0

```
with open("./data/paper.txt", "r", encoding="utf-8") as input_file:
```

```
    for line in input_file:
```

```
        print(line)
```

```
        line_count =line_count+1
```

```
print("전체 글의 줄수는 {0}".format(line_count))
```

파일 입출력

◆ 파일 출력

```
f = open("./data/sample.txt", 'w', encoding="utf-8")
for i in range(1, 11):
    data = "%d번째 내용입니다.\n" % i
    a=f.write(data)
    print("쓴 글의 글자수 {0}".format(a))
f.close()
```

◆ 파일 내용 추가

```
f = open("./data/sample.txt", 'a', encoding='utf-8')
f.write('새로운 내용추가\n')
f.close()
```

```
◆ with open('./data/sample.txt', 'a', encoding='utf-8') as file_write:
    for index in ['aaa', 'bbb', 'ddd']:
        file_write.write("새로운 내용{0}\n".format(index))
```

표준 라이브러리

◆ datetime : 날짜와 시간을 조작하는 클래스

```
import datetime  
print(datetime.date.today())  
print(datetime.datetime.now())  
  
n= datetime.datetime.now()  
d=n.strftime("%Y %m %d")  
print(d)
```

<https://docs.python.org/ko/3.8/library/index.html>

표준 라이브러리

◆ Counter : 해쉬객체 카운트를 위한 dict의 서브 클래스

```
from collections import Counter
data="사과 배 복숭아 딸기 포도 배 복숭아 포도"
d=data.split()
wcount=Counter()

for index in d:
    wcount[index] += 1

print(wcount)
```

표준라이브러리

◆ Defaultdict

◆ from collections import defaultdict

```
s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]
d = defaultdict(list)
for k, v in s:
    d[k].append(v)

slist=sorted(d.items())
print(slist)
```

⇒ 처리결과

```
[('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]
```