파이썬 함수

Function

1. 파이썬 함수종류
2. 사용자 정의 함수
3. 함수의 호출과 흐름
4. 함수의 인자와 반환값
5. 함수를 인자로 전달하기
6. 기본 함수(Built-in 함수)
7. 라이브러리(패키지) 함수

파이썬 함수종류

함수의 세 가지 종류

- 사용자 정의 함수
  - 사용자가 자신이 필요로 하는 기능을 수행하는 함수를 작성한 함수들
- 기본 함수 혹은 built-in 함수
  - 기본 함수는 파이썬의 실행과 동시에 사용할 수 있는 함수들.
    - len(), sum(), min(), max(), sorted()
    - Id(), type()
- 라이브러리 혹은 패키지 함수
  - 해당 라이브러리를 포함한 후에 사용할 수 함수들.
    - Import 라이브러리 : import time
    - From 라이브러리 import 함수명 as 축약 : os.getcwd()

파이썬 함수종류

함수를 작성하는 이유

- 특정의 기능을 수행하는 코드들을 하나의 묶음으로 사용
- 재사용성을 높이고 코드의 통일된 관리를 하기 위해 함수를 작성함.

- 함수 문법

```
def  함수명 ([인자1, 인자2, ...]):
    수행할 문장들
return 반환값
```

사용자 정의 함수

- 함수의 인자전달 시 값 전달과 객체 전달 차이 이해

```
def call_by_value(num, mlist) :
    num = num + 1
    mlist.append("add 1")

num = 10
mlist = [1,2,3]

print(num, mlist)
call_by_value(num, mlist)
print(num, mlist)
```

사용자 정의 함수

함수의 인자와 반환값

- 함수를 실행할 때 외부로부터 인자를 받아서 처리할 수 있다.
- 외부로부터 넘어온 값은 함수 내부에서 자유롭게 사용이 가능하다.
- 함수는 작업을 마친 후 호출한 지점으로 돌아갈 때 반환값을 되돌려 줄 수 있다
  .

- return 문
  - return # 제어를 되돌리고 None 값을 반환
  - return 반환값 # 제어를 되돌리고 반환 값을 반환

사용자 정의 함수(인자)

- 정의되지 않은 인자 전달
  정의되지 않는 인자 전달이 가능하며, 이는 dic 타입으로 전달됨
  일반인자->가변인자->정의되지 않은 인자 순으로 전달되어야 함

```
def circle_area( radius, *pi, **info ):
    for item in pi :
        area = item * (radius ** 2 )
        print("반지름 : " , radius, "PI : ", item, "면적 : ", area)

    for key in  info :
        print(key , " : ", info[key])



circle_area(3, 3.14, 3.1456, 3.141592)
print()
circle_area( 3, 3.14, 3.1456, line_color="red", area_color="green )
```

사용자 정의 함수

함수의 인자로 함수 전달하기(고차함수)

round함수 : 숫자 자릿수 처리함수

```
def circle_area(radius, print_format):
  area = 3.14 * (radius ** 2)
  print_format(area)

def precise_low(value):
    print("결과값:", round(value, 1)) # 반올림해서 소수점 한자리까지 출력

def precise_high(value):
    print("결과값:", round(value, 2)) # 반올림해서 소수점 둘째자리까지 출력

circle_area(3, precise_low)
circle_area(3, precise_high)
```

함수의 스코프

- Builtin scope : 파이썬이 제공하는 내장모듈 공간
- Global scope : 함수안에 포함되지 않은 전역 공간
- Enhanced local scope : 파이썬은 함수를 중첩가능함. 중첩된 영역의 공간
- Local scope : 함수 내에 정의된 지역 공간

함수의 스코프

```
g_val = 3
def foo():
        l_val= 2
foo()
```

- g_val은 프로그램 전역공간에서 사용가능
- l_val은 foo함수 안에서만 사용가능
- foo 함수 밖에서 l_val을 사용 한다면?

```
g_val = 3
def foo():
    g_val = 2
foo()
print g_val
```

- foo함수 밖의 g_val은 전역변수
- foo함수 안의 g_val은 지역변수 (l-value)
- 출력 결과는?

```
g_val = 3
def foo():
    l_val = g_val
    print l_val
foo()
```

- foo 함수 안의 g_val은 전역변수 (r-value)
- 따라서 foo함수 안에서 사용 가능
- 출력 결과는?

함수의 스코프

```
g_val = 3
def foo():
    global g_val
    g_val  = 2
foo()
print g_val
```

- foo 함수 안에서 g_val의 값이 l-value에 오면 지역변수
- foo 함수 안에서 전역변수 g_val의 값을 바꾸려면?
  - global 키워드를 사용해야 함
  - global 키워드 이후로는 g_val을 전역변수로 인식함
- 출력 결과는?

- 전역변수를 남발하면?
  - 문제 발생시 젂역변수의 값을 참조하는 모든 코드를 디버깅 해야함
  - 전역역 변수를 많이 참조 할수록 디버깅은 어려워짐
  - 함수 안의 범위만 쉽게 디버깅 하기 위해서 지역변수를 사용을 권고함.

함수의 스코프 – 예제

```
x = "global“
def foo():
   print("x inside :", x)
foo()
print("x outside:", x)
```

```
x = 5
def foo():
   x = 10
   print("local x:", x)
foo()
print("global x:", x)
```

```
x = "global“
def foo():
   x = x * 2   # 오류 발생
   print(x)
foo()
```

```
def foo():
   y = "local“
foo()
print(y) #오류 발생
```

함수의 스코프

- **nonlocal vs global**

```
x = 'global'
def outer():
    x = "local"

    def inner():
        nonlocal x
        x = "nonlocal"
        print("inner:", x)

    inner()
    print("outer:", x)


outer()
print("global : ", x)
```

```
def scope_test():
    def do_local():
        spam = "local spam"
        print("dolocal", spam)
    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam”
    def do_global():
        global spam
        spam = "global spam”
    spam = "test spam"
    do_local()
    print("After local assignment:", spam
)
    do_nonlocal()
    print("After nonlocal assignment:", s
pam)
    do_global()
    print("After global assignment:", spa
```

람다(Lamda) 함수 == 익명함수

Lambda 인자들 : 표현식

- 아주 작으며 이름이 없는 함수
- 문법적으로는 하나의 표현식만 사용되어야 함

```
def hap(x, y):
  return x + y


print(hap(10, 20))
print(( lambda x,y: x + y)(10, 20) )
```

람다(Lamda) 함수 == 익명함수

람다 함수를 인자로 전달

```
pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]

def shortKey(pair) :
    return pair[1]


pairs.sort(key=shortKey)
# pairs.sort(key=lambda pair: pair[0])    lambdar로 함수 전달

print(pairs)
```

람다(Lamda) 함수 == 익명함수

함수를 반환하는 함수

```
def make_incrementor(n):
    print("n", n)
    return lambda x: x + n

f = make_incrementor(42)  # f 함수를 생성함

print(make_incrementor(0))
print(f)
print(f(0))

print(f(1))

print(make_incrementor(65)(1))
```

람다(Lamda) 함수 == 익명함수

활용 예제

map()

map(함수, 리스트)
리스트로부터 원소를 하나씩 꺼내서 함수를 적용시킨 다음,
그 결과를 새로운 목록을 map 객체로 반환하는 함수

```
arr = [1,2,3,4,5]
brr = map(lambda  x : x**2, arr)
print(brr)
crr = list(brr)
print(crr)
```

filter()

filter(함수, 리스트)
리스트로부터 원소를 하나씩 꺼내서 함수를 적용시킨 다음,
조건에 만족하는 원소만 새로운 목록으로 반환하는 함수

```
arr = [1,2,3,4,5,6,7,8,9]
brr = filter(lambda x :x%3 == 0, arr)
print(brr)
#print(next(brr))
for item in brr :
    print(item, end="")
```