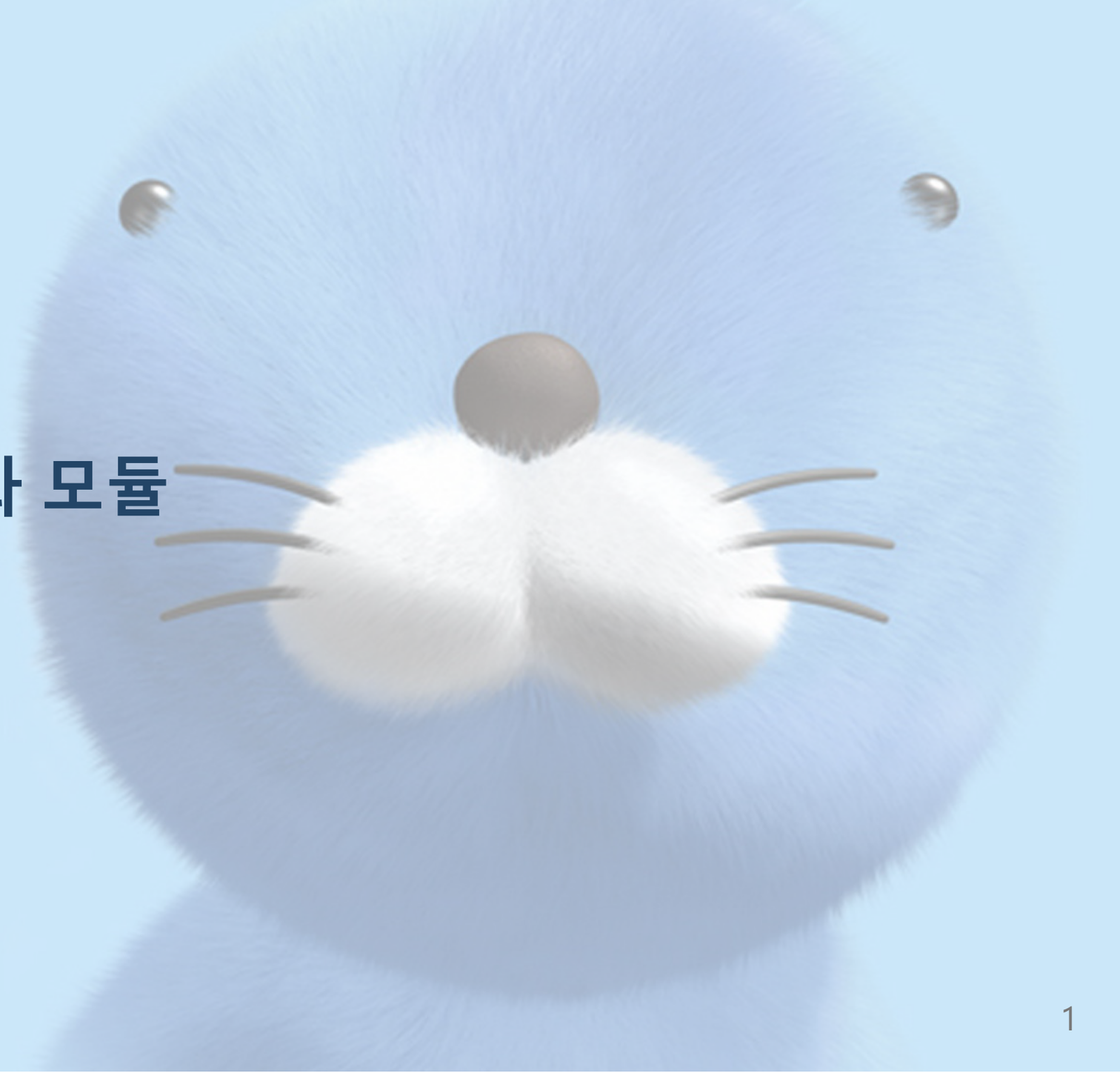
# Python 신입 SIG: 함수와 모듈

Oct 8th/2017



# 함수

- 프로그램에서 함수는 특정한 작업을 수행하는 코드의 모임을 의미함.
- 함수는 프로그램의 내부에서 또는 외부 라이브러리에서 정의될 수 있다.
- 함수는 경우에 따라서 인자를 받거나 결과값을 반환할 수도 있다(필수는 아님).
- 프로그램에서 반복되는 부분을 함수로 처리하는 것이 좋다.

```
def <Name of the function>(<Parameter(s)>):
    ...
    <Some codes>
    ...
    return <Value to be returned> # optional
# Call
<Function name>(<Params>)
```

Python 3.6.2 Documentation - Defining Functions

# 예시

```
def hello(): #정의
    print("Hello, World!")

hello() # 호출
```

```
Hello, World!
```

# 매개변수

- 함수를 실행시킬 때 값을 받기 위해 사용.
- 함수를 정의할 때 사용한 것을 매개변수라 하고, 실행시 넘긴 값을 실행인자라고 함.
- 매개변수는 함수 이름 뒤의 괄호 안에 쉼표로 구분하여 넣음.
- 실행인자와 매개변수의 수가 일치해야 함.

```
def printAbs(val):
# 매개변수 val ^
    if val<0:
        val= -val
    print(val)

printAbs(-2.12)
# 실행인자  ^
```

```
2.12
```

## 기본값 지정

- 기본값을 지정하면 받아야 하는 인자 수보다 적은 수를 받을 경우 기본으로 지정된 값을 인자로 받아들인다.
- 기본값을 지정한 매개변수들은 반드시 기본값이 지정되지 않은 것들보다 뒤에 있어야 한다.
- `<param>=<default>` 와 같이 사용

```
def rootCalc(x,err=0.0001,maxstep=100): # OK
        # 생략 ...
    return rt
def rootCalc2(err=0.0001,x,maxstep=100): # Wrong
        # 생략 ...
    return rt

rootCalc(2.00,0.01,50)
rootCalc(2.00,0.001) # rootCalc(2.00,0.001,100)과 같다.
rootCalc(2.00) # rootCalc(2.00,0.0001,100)과 같다.
```

Python 3.6.2 Documentation - Default Argument Values

## Keyword Arguments

- 호출 시 매개변수의 이름을 사용하여 호출할 수 있다.
- Non-keyword argument는 반드시 keyword argument 전에 나와야 한다.
- `<kwarg>=<value>` 와 같이 사용함.

```
def rootCalc(x,err=0.0001,maxstep=100): # OK
        # 생략 ...
    return rt
rootCalc(2,maxstep=30) # == rootCalc(2,0.0001,30)
rootCalc(2,maxstep=20,0.001) # Wrong
rootCalc(err=0.000001,x=2.0,maxstep=1000) # OK
```

Python 3.6.2 Documentation - Keyword Arguments

## 정해지지 않은 갯수의 값 받기

- Non-keyword argument는 `*args` , keyword argument는 `**kwargs` 형태의 매개변수를 사용하여 받을 수 있다.
- Non-keyword argument들은 tuple, keyword argument들은 dictionary(keyword를 키로 하는)로 받아진다.
- `*args` , `**kwargs` 의 이름은 관습적인 것으로, 아무 이름이나 사용 가능하며 이들 매개변수 뒤에는 반드시 keyword argument만 올 수 있다.

```
def sumAll(useAbs,*args):
        total=0
    for x in args:
        if useAbs:
            total+=abs(x)
        else:
            total+=x
    return totsl
```

# 값의 반환

- `return`을 사용해 결과값을 돌려줄 수 있다.
- 결과값은 없거나 1개뿐이다.
- 모든 자료형이 가능하다.
- 여러 개의 결과값은 tuple, list 등으로 묶어 하나로 만들어 반환한다.

```
def sum(a,b):
        return a+b # a+b를 돌려줌

x = sum(1.12+6.24) # 받은 값을 x에 저장
print(x)
```

```
7.36
```

# 모듈

변수, 함수, 클래스 등을 모아 다른 프로그래에서도 사용할 수 있도록 한 파일

- `import <module>`으로 load
- 모듈 이름이 쓰기 복잡할 경우 `import <module> as <name>`로 다른 이름으로 모듈을 불러올 수 있다.
  - `<module>.<item>`으로 구성요소 사용
- `from import <item(s)>으로 특정 요소만 가져와 모듈 이름 없이 부를 수 있음
  - `<item>`으로만 부른다.
  - `from <module> import *`로 모든 구성요소를 가져올 수 있다.
  - 이름이 충돌할 수 있으므로 꼭 필요한 경우에만 사용.

```
import antigravity
antigravity.geohash(37.421542, -122.085589, b'2005-05-26-10458.68
```

```
import antigravity as AG
AG.geohash(37.421542, -122.085589, b'2005-05-26-10458.68')
```

```
from antigravity import geohash
geohash(37.421542, -122.085589, b'2005-05-26-10458.68')
```

Python 3.6.2 Documentation - Modules

## Example

`math` : 수학 관련 모듈

`random` : 무작위 관련 모듈

```
import random
items = ['피자','치킨','짜장면','도시락']
print(random.choice(items))
```

```
치킨
```

# 모듈의 작성

- 모듈을 구성할 함수, 변수 등의 정의를 작성하고, 파일을 `<modulename>.py` 로 저장한다.
- 기존 모듈과 동일하게 import하고 사용한다.
- 단, 모듈을 사용할 파일과 같은 directory에 있어야 한다.

```
# Fibonacci numbers module
# fibo.py
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

## Example - `datetime`

- 날짜/시간 관련 모듈
- `datetime.date` (날짜), `datetime.time` (시간(날짜 상관없이)), `datetime.datetime` (날짜와 시간), `datetime.timedelta` (시간 차이), `datetetime.tzinfo` , `datetime.timezone` (시간대)의 클래스가 있음.
- 여기서는 `datetime.datetime` 과 `datetime.timedelta` 에 대해서만 알아볼 것임.

  Python 3.6.2 Documentation - `datetime`

# Example - `datetime`

**`datetime.datetime`**

- `datetime`에는 `year`, `month`, `day`, `hour`, `minute`, `second`, `microsecond`의 시간 관련 attribute가 있다.
- `<datetime object>.now()`는 지금 날짜, 시간을 담은 datetime 객체를 반환한다.
- `<datetime object>.replace`메서드를 통해 값을 변경할 수 있다.
- `datetime`끼리의 차는 `timedelta` 객체를 반환한다.

```
import datetime
time_now = datetime.datetime.now() # 지금 시간
time_9pm = time_now.replace(hour=21) # 값 변경
until_9pm = time_9pm-datetime.datetime.now() # 시간 차이
```

## Example - `datetime`

**`datetime.timedelta`**

- 시간 사이의 차이를 나타냄.
- 생성시 `days`, `seconds`, `microseconds`, `miliseconds`, `minutes`, `hours`, `weeks`를 인자로 받는다(기본값=0).
- `datetime`에 더하거나 뺄 수 있다.