Python 기초 강의

5 Weeks

Cyber-Physical Systems Security

순천향대학교 정보보호학과

김준원

2018.05.14

Agenda

❖내장 함수 및 외장 함수

❖모듈 (Module)과 패키지

내장 함수 및 외장 함수

내장 함수(Internal function)

외장 함수(External function)

내장 함수 활용 및 문자열 함수

❖ 내장 함수 (Internal function)

- ✓ Import를 필요로 하지 않는 함수 (모듈을 import할 필요가 없는 함수)
- ✓ 이미 만들어진 함수

❖ abs(value)

- ✓ Value가 숫자일 때, 그 값의 절대값을 돌려주는 함수

```
>>> abs(4)
4
>>> abs(-4)
4
>>> abs(-4.98)
4.98
```

내장 함수 활용 및 문자열 함수

- all(iter)
  - ✓ 반복 가능한 자료형(Iterable)를 인자로 받으며, iter가 모두 참이면 True, 아니면 False를 반환함
  - ✓ Iterable: for문으로 값을 출력할 수 있는 자료형
    - ✓ ex) List, Tuple, String, directory, Set

```
>>> all([1, 2, 3])
True
>>> all([1, 2, 3, 0])
False
>>> all((1, 4, 8))
True
```

내장 함수 활용 및 문자열 함수

❖ any(iter)

✓ 반복 가능한 자료형(Iterable)를 인자로 받고 iter가 하나라도 참이면 True, 아니면 False를 반환함

```
>>> any([1,2,3,0])
True
>>> any((0, “”, 4))
True
>>> any([0, “”])
False
```

❖ dir(object)

✓ object가 가지고 있는 변수와 함수를 보여줌

```
>>> dir([1,2,3]) # List 타입
[‘append’, ‘count’, ‘extend’, ‘index’, …]
>>> dir({‘1’:’a’}) # directory 타입
[‘clear’, ‘copy’, ‘get’, ‘has_key’, …]
```

내장 함수 활용 및 문자열 함수

❖ divmod(x,y)

- ✓ 2개의 숫자를 인자로 받으며 x를 y로 나눈 몫과 나머지를 튜플형으로 반환함

```
>>> divmod(7,3)
(2,1)
>>> divmod(1.3, 0.2)
(6.0, 0.09999999999999998)
```

❖ enumerate(iter)

- ✓ Iterable형을 가진 값을 인자로 받으며 인덱스 값과 그 인덱스 값을 포함한 값을 반환함
- ✓ for~each에서 유용하게 사용됨

```
for index, value in enumerate(['one', 'two', 'three'])
    print("%d번째 값: %s" % (index, value))
…
0번째 값: one
1번째 값: two
2번째 값: three
```

내장 함수 활용 및 문자열 함수

❖ eval(expression)

✓ 실행 가능한 문자열(1+2나 'hi'+'python'과 같은 식)을 인자로 받아 실행한 결과값을 반환

```
>>> eval('12+34')
46
>>> eval("'hi' + ' python'")
'hi python'
>>> eval('divmod(4, 3)')
(1, 1)
```

❖ fliter(function, iter)

✓ 각각 함수와 iterable형을 인자로 받으며 function을 통해 리턴값이 참인 것만 묶어서 반환

```
def func(x):
    return x > 0

print(list(filter(func, [1, -3, 5, 7, -4])))
…
[1,5,7]
```

내장 함수 활용 및 문자열 함수

❖ isinstance(object, class)

✓ 각각 인스턴스 변수와 클래스를 인자로 받으며, object가 class의 인스턴스인지 판단하여 참이면 True, 아니라면 False를 반환함

```
>>> a = 4
>>> isinstance(a, int)
True
>>> isinstance(a, str)
False
>>> a = {1, 3, 5}
>>> isinstance(a, list)
False
>>> isinstance(a, set)
True
```

내장 함수 활용 및 문자열 함수

❖ len(s)

✓ s의 길이 또는 요소의 전체 개수를 반환함

```
>>> len(“python”)
6
>>> len([1,3,5])
3
>>> len(1, ‘b’)
2
```

❖ list(v)

✓ Iterable형을 인자로 받으며 리스트로 만들어 반환함

```
>>> list(“python”)
[‘p’, ‘y’, ‘t’, ‘h’, ‘o’, ‘n’]
>>> list((2,4,6,8))
[2,4,6,8]
>>> list({1,2,3,4})
[1,2,3,4]
```

내장 함수 활용 및 문자열 함수

❖ map(function, iter)

- ✓ 함수와 Iterable형을 인자로 받으며, iter는 function을 걸쳐 그 function을 통한 실행 결과를 바탕

```
>>> def func(x): return x*2
…
>>>list(map(two_times, [1,2,3,4]))
[2,4,6,8]
>>>list(map(lambda func: x*2, [1,2,3,4])) # 위와 같음
[2,4,6,8]
```

❖ max(iter), min(Iter)

- ✓ Iterable형을 인자로 받으며, max 함수는 최대값을, min 함수는 최소값을 반환함

```
>>> min([1,2,4,5])
1
>>> max('python')
'y'
```

내장 함수 활용 및 문자열 함수

❖ type(object)

- ✓ object의 타입(클래스 형태)을 반환함

```
>>> type(2)
<class ‘int’>
>>> type([2, 4])
<class ‘list’>
```

❖ tuple(iter)

- ✓ Iterable형을 인자로 받으며 튜플 형태로 바꿔 반환함

```
>>> tuple(“abc”)
(‘a’,’b’,’c’)
>>> tuple([2, 4, 6, 8])
(2, 4, 6, 8)
```

내장 함수 활용 및 문자열 함수

❖ range(start, end, step)

- ✓ Int형을 인자로 받으며 해당하는 범위의 값을 반복 가능한 객체로 반환함
- ✓ 인수가 1개라면: 시작 값이 0부터 시작함
- ✓ 인수가 2개라면: 시작 숫자와 끝 숫자를 나타내며, 끝 숫자까지 범위가 포함되지 않음
- ✓ 인수가 3개라면: 숫자 사이의 거리를 조절함

```
>>> list(range(6))
[0, 1, 2, 3, 4, 5]
>>> list(range(1, 7))
[1, 2, 3, 4, 5, 6 ,7]
>>> list(range(0, -7))
[0, -1, -2, -3, -4, -5, -6]
>>> list(range(1, 10, 3))
>>> [1, 4, 7]
```

내장 함수 활용 및 문자열 함수

❖ 문자열 함수

✓ str(String)형을 가진 인스턴스에 제공되는 내장 함수

❖ format

✓ 지정된 형식이 포함된 문자열을 값을 변환한 문자열로 바꿔주고 반환하는 함수

✓ 번호를 활용하여 넣기

```
>>> “Hello, {0}”.format(‘python’)
‘Hello, python’
>>> “{0}, {1}”.format(‘hello’, ‘python’)
‘Hello, python’
```

✓ 이름을 활용하여 넣기

```
>>> “Hello, {lang}! My name is {name}”.format(lang=‘python’, name=‘bob’)
```

✓ 번호 + 이름을 활용하여 넣기

```
>>> “Hello, {0}! My name is {name}”.format(‘python’, name=‘bob’)
```

내장 함수 활용 및 문자열 함수

❖ 정렬하기

✓ 왼쪽 & 오른쪽 & 가운데 정렬

```
>>> “{0:<10}”.format(“hi”) # index:<n = index번째는 n자리 만큼 자리를 만들고 왼쪽으로 정렬하도록 지정함
‘hi        ‘
>>> “{0:>10}”.format(“hi”) # index:>n = 위 기능과 같으며 오른쪽 정렬
‘        hi’
>>> “{0:^10}”.format(“hi”) # index:^n = 위 기능과 같으며 가운데 정렬
‘    hi    ‘
```

✓ 공백에 문자 채워 넣기

```
>>>”{0^=10}”.format(‘hi’)
‘====hi====‘
>>>”{0:!<10}”.format(‘hi’)
‘hi!!!!!!!!’
```

✓ 소수점 표현하기

```
>>>”{0:6.3f}”.format(3.42)
‘  3.420’
```

내장 함수 활용 및 문자열 함수

❖ f 문자열 formatting

- ✓ format 함수와 역할은 비슷하나 표현식, Directory를 추가로 사용할 수 있음
- ✓ Format 함수를 사용했을 때보다 더 간략하게 형식 지정을 할 수 있음
- ✓ Python 3.6이상에서 사용 가능하며 하위 버전은 호환되지 않음

```
>>> name = ‘bob’
>>> age = 31
>>> f’나의 이름은 {name}입니다. 나이는 {age}입니다.’
‘나의 이름은 bob입니다. 나이는 31입니다.’
# 표현식 지원
>>> age = 30
>>> f’내년이면 난 {age+1}살입니다.‘
‘내년이면 난 31살입니다.’
# 딕셔너리
>>> dic = {‘name’:’bob’, ‘age’:31}
>>> f’나의 이름은 {d[‘name’]}입니다. 나이는 {d[‘age’]}입니다.’
‘나의 이름은 bob입니다. 나이는 31입니다.’
```

내장 함수 활용 및 문자열 함수

❖ count(chr)

✓ 문자를 인자로 받으며, 문자 chr의 개수를 반환함

```
>>> a = ‘hobby’
>>> a.count(‘b’)
2
```

❖ find(chr)

✓ 문자를 인자로 받으며, 문자 chr이 처음으로 나온 위치(인덱스 번호)를 반환함

✓ 찾지 못했을 경우에는 -1을 반환함

```
>>> a = “Python is best choice”
>>> a.find(‘b’)
10
>>> a.find(‘k’)
-1
```

내장 함수 활용 및 문자열 함수

❖ index(chr)

- ✓ Find 함수와 같음
- ✓ 찾지 못했을 경우에는 -1을 반환하는 것이 아닌 ValueError가 발생함

```
>>> a = “Python is best choice”
>>> a.index(‘b’)
10
>>> a.index(‘k’)
Traceback (most recent call last): File "<stdin>", line 1, in <module>
ValueError: substring not found
```

❖ join(str)

- ✓ 문자열을 인자로 받으며, str 문자 사이 사이에 해당 인스턴스 문자를 끼워 넣음

```
>>> split=‘,’
>>> split.join(‘abcde’)
‘a,b,c,d,e’
```

내장 함수 활용 및 문자열 함수

❖ replace(target, to)

✓ 문자열을 인자로 받으며, target 문자열을 to 문자열로 바꾼 후 반환함

```
>>> a= “python is Best Choice”
>>> a.replace(‘Best Choice’, ‘good’)
‘python is good
```

❖ split(chr)

✓ 문자열를 인자로 받으며, 문자열 촉 단위로 나눠서 리스트로 반환함

```
>>> a=“hello world python!”
>>> a.split() # 인자값이 없으면 기본값으로 스페이스
[‘hello’, ‘world’, ‘python!’]
>>> a = ‘a:b:c:d’
>>> a.split(‘:’)
[‘a’,’b’,’c’,’d’]
```

외장 함수

❖ 외장 함수 (External Function)

- ✓ 파이썬 라이브러리 내 이미 정의되어 있는 유용하게 사용할 수 있는 함수
- ✓ 대표적으로 sys, os 모듈이 있음 (이 외에도 무수히 많음, 직접 찾아보는 것을 권함)
- ✓ import를 이용하여 모듈을 가져와 사용함

❖ os

- ✓ 환경 변수나 디렉터리, 파일 등의 OS 자원을 제어할 수 있게 해주는 모듈

❖ sys

- ✓ 파이썬 인터프리터가 제공하는 변수들과 함수들을 직접 제어할 수 있게 해주는 모듈

외장 함수

❖ sys.argv

✓ 프로그램을 실행할 때 첨부 값으로 넣어준 인자 값들을 모아둔 변수

```
# test.py
import sys
print(sys.argv)

# Command Prompt
C:/Users/CPSS>python test.py you need to python
['test.py', 'you', 'need', 'to', 'python']
```

❖ sys.exit()

✓ 프로그램을 중단함

외장 함수

❖ sys.path

✓ 프로그램을 실행할 때 첨부 값으로 넣어준 인자 값들을 모아둔 변수

✓ 경로에 상관없이 어디서나 호출 가능

```
>>> import sys
>>> sys.path
 ['', 'C:\\Windows\\SYSTEM32\\python36.zip', 'c:\\Python36\\DLLs',
'c:\\Python36\\lib', 'c:\\Python36', 'c:\\Python36\\lib\\site-
packages']
```

❖ 모듈 경로 추가하기

✓ sys.path는 list형이므로 append 내장 함수를 사용하여 추가

외장 함수

❖ os.environ(v)

✓ 시스템 환경 변수 값을 가져오는 함수

```
>>> import os
>>> os.environ
environ({'PROGRAMFILES': 'C:\\Program Files', 'APPDATA': … skip …})
```

❖ 디렉토리 위치

✓ os.chdir(path) : 현재 디렉토리 위치 변경

✓ os.getcwd() : 현재 디렉토리

❖ os.system(command)

✓ 문자열을 인자 값으로 받으며, 시스템에 내장된 프로그램이나 명령어를 호출함

모듈 (Module)과 패키지

모듈 (Module)

패키지(Package)

모듈(Module)

❖ 모듈(Module) `import 이름`

- ✓ 함수나 변수, 또는 클래스들을 모아 놓은 파일
- ✓ 다른 Python 프로그램에서 불러와 사용할 수 있음
- ✓ import 예약어를 사용하여 모듈을 불러와 사용

❖ 모듈 내 함수 호출하기

```
# my_module.py
def print_str(str):
    print(“hello! %s” % str)

>>> import my_module
>>> my_module.print_str(‘world!’)
```

모듈(Module)

❖ 모듈 내 특정 함수만 가져와 사용하기

```
# my_module.py
def print_str(str):
    print(“hello! %s” % str)

def print_other(str):
    print(“Your name: %s” % str)

def test():
    pass
…

>>> from my_module import print_str                # my_module 모듈에서 print_str 함수만 가져와 사용
>>> print_str(‘world!’)
hello world!
>>> from my_module import print_str, test          # my_module 모듈에서 print_str, test 함수와 가져와 사용
>>> test()                                         # pass만 있기에 아무것도 실행할 것이 없음
>>> print_other(‘world’)
NameError: name ‘print_other’ is not defined       # my_module 모듈에서 print_other 함수를 가져오지 않았음
>>> from my_module import *                        # my_module 모듈에 있는 모든 함수를 가져와 사용
```

모듈(Module)

- __name__
  - ✓ 해당 모듈의 이름을 저장해둔 Python 내부 변수
  - ✓ 해당값은 모듈의 파일명과 같음 (ex: mod1.py => __name__="mod1")
  - ✓ 필요성
    - ✓ Import의 역할이 모듈을 가져옴과 동시에 그 파일을 실행하는 것이므로 모듈이 실행되는 것을 차단할 때 사용함
    - ✓ 단순 py 파일을 실행하는 것과 import을 위한 모듈 역할로 나눌 때 참고로 하여 사용함

모듈(Module)

❖ __name__을 이용하여 용도 나누기 (사용하기 전)

```
# mod1.py 소스 코드
def sum(a, b):
  return a+b

def safe_sum(a, b):
  if type(a) != type(b):
    print("더할수 있는 것이 아닙니다.")
    return
  else:
    result = sum(a, b)
    return result

print(safe_sum('a', 1))
print(safe_sum(1, 4))
print(sum(10, 10.4))
```

```
# command prompt (명령 프롬프트)
C:\Users\CPSS>python mod1.py
더할 수 있는 것이 아닙니다.
None
5
20.4
```

정상 작동

```
# command prompt (명령 프롬프트)
C:\Users\CPSS>python
>>> import mod1
더할 수 있는 것이 아닙니다.
None
5
20.4
```

Import했을 때 파일이 실행되어 출력 결과까지 나옴

모듈(Module)

❖ __name__을 이용하여 용도 나누기 (사용하기 후)

```
# mod1.py 소스 코드
def sum(a, b):
  return a+b

def safe_sum(a, b):
  if type(a) != type(b):
    print("더할수 있는 것이 아닙니다.")
    return
  else:
    result = sum(a, b)
    return result
if “__name__” == “__main__”:
     print(safe_sum('a', 1))
     print(safe_sum(1, 4))
     print(sum(10, 10.4))
```

```
# command prompt (명령 프롬프트)
C:\Users\CPSS>python mod1.py
더할 수 있는 것이 아닙니다.
None
5
20.4
```

정상 작동

```
# command prompt (명령 프롬프트)
C:\Users\CPSS>python
>>> import mod1
>>> mod1.sum(2,4)
6
>>> mod1.safe_sum(‘a’,2)
더할수 있는 것이 아닙니다.
```

작업 모듈(name)이 main(current)이 아니므로 실행되지 않음

패키지(Package)

❖ 패키지(Package)

- ✓ 모듈을 구조적으로 나누어 주는 것
- ✓ 여러 개의 모듈을 패키지를 이용하여 기능들을 구조화해 보여줄 수 있어 관리에 용이함
- ✓ 패키지 관리를 위한 __init__.py이라는 특별한 파일이 사용됨

```
game/
  __init__.py
  sound/
      __init__.py
      echo.py
      wav.py
  graphic/
      __init__.py
      screen.py
      render.py
  play/
      __init__.py
      run.py
      test.py
```

패키지(Package)

❖ __init__.py의 용도

- ✓ 해당 디렉터리가 패키지의 일부임을 알려주는 역할을 함
- ✓ 해당 파일이 없으면 해당 디렉토리의 모듈을 읽어올 수 없음 (패키지로 인식하지 않음)

마침

궁금하신 점이나 질문이 따로 있다면?

개인 이메일 ruskonert@gmail.com

동아리 이메일 support@cpss.network

동아리 저장소 https://github.com/CPSSOpenSource

과제 결과 확인 https://github.com/CPSSOpenSource/Python-Report-Complie

강의를 들어 주셔서 감사합니다.