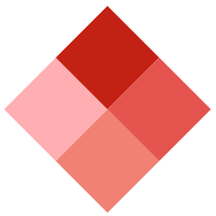


파이썬을 활용한 업무자동화

3회차: 프로그래밍 언어에 대한 이해

목차

3회차: 파이썬 필수 문법 (2)

- 복습
- 함수
  - 왜 필요할까?
  - 함수의 기본
  - 반환하는 함수? 반환하지 않는 함수?
  - 스코프 (변수의 효력 범위)
  - 파이썬 내장함수
- 클래스
  - 왜 필요할까?
  - 클래스의 기본
  - 클래스의 활용과 생성자

복습

지난 강의 때 뭐했지..?

제어문: if, for, while

복습

지난 강의 때 뭐했지..?

- 제어문: if, for, while

```
emp = []

emp.append({'name':'taehwa', 'age':30, 'position':'manager'})
emp.append({'name':'yongseong', 'age':28, 'position':'intern'})
emp.append({'name':'jungeun', 'age':32, 'position':'ceo'})

person_count = 0
for person in emp:
    print(person)
    if person['position'] == 'ceo':
        print('pass ceo')
        continue
    if person['age'] >= 30:
        person_count = person_count + 1

print(person_count)
```

함수

왜 필요할까?

함수란??

- 입력값을 받아, 특정 작업을 수행한 후에 결과값을 주는 것!
- 하지만! 입력값이 없을 수도, 결과값이 없을 수도 있음

예시: 문자열 내장함수

```
company = 'fastcampus'
a_count = company.count('a')
print(a_count)
```

함수를 호출한다
└ 함수

입력값을 받아, 특정 작업을 수행한 후에 결과값을 주는 것

함수

왜 필요할까?

함수가 없으면?

- 코드를 보는 사람도 불편하고 개발 하는 사람도 불편..
- 효율이 너무 떨어진다!
  - 같은 내용이 반복된다..

예시: 함수가 없으면…

- 이메일을 보내야하는 상황이 여러 경우라고 가정
- 이메일을 보내기 위해 코드 10줄을 작성해야 한다고 가정
- 코드 10줄을 필요할때마다 복붙을 해야하는가!!
  - 복붙을 안하려고 코딩하는데 왜 코딩하면서 복붙을 하는가..

이메일 보내는 함수를 만들고 함수만 호출!

함수

함수의 기본

함수의 모습

```
def 함수이름(입력값 변수, 입력값 변수, ...):
    <실행할 코드>
    <실행할 코드>
    <실행할 코드>
```

함수는..

- 함수 호출이 되기 전까지 함수 안에 있는 문장은 수행이 안됨!
- 함수는 호출 되기 전에 먼저 만들어져야 함!!
- 입력값은 함수안에서 변수로 사용됨

제어문과 마찬가지로 공백 중요 !!

함수

함수의 기본

함수의 동작과정

```
def print_name():
    print('-'*15)
    print('My name is \"taehwa\"')

print("Hello fastcampus")
print_name()
print_name()
print_name()
print_name()
```

```
alghost:3rd Alghost$ python3 008.py
Hello fastcampus
---------------
My name is "taehwa"
---------------
My name is "taehwa"
---------------
My name is "taehwa"
---------------
My name is "taehwa"
```

함수

반환하는 함수

반환하는 함수란?

- 함수를 호출한 곳에 결과값을 전달하는 함수
- 따라서 함수 내에서 실행한 문장들의 결과값이 필요할 때 사용

반환은 어떻게..

- return 값
- 함수는 return을 만나면 끝나버림!

예제: 덧셈

- 입력값 2개를 받아 덧셈 결과를 반환
- 변수를 써서 받아도, 그대로 써도 됨!

```python
def my_sum(a, b):
    return a+b

res = my_sum(10, 20)
print(res)

print(my_sum(30, 40))

my_sum(50, 60)
```

```
alghost:3rd Alghost$ python3 009.py
30
70
```

함수

반환하지 않는 함수

반환하지 않는 함수란?

- 함수를 호출한 곳에 아무 결과값 전달 없이 문장만 수행
- 함수내 문장을 수행한 결과가 호출한 곳에서 필요없을 때 사용

예제: 특정 문자열을 제외하고 출력하기!

- 문자열을 입력받아 'skip'이 포함되어있다면 출력하지 않기

return 사용 가능!

- 아무 값 없이 return 만 사용!
- 함수를 빠져나갈 때 사용가능

함수

반환하지 않는 함수

- 반환
  - 함수
  - 함수
- 예제
  - 문자열
  - (욕설
- retu
  - 아무
  - 함수를 빠져나갈 때 사용가능

용

```
def print_without_skip(string):
    if 'skip' in string:
        print('Rejected')
        return
    else:
        print(string)

    print('-'*10)

user_input = ''
while user_input != 'quit':
    user_input = input('Input: ')
    print_without_skip(user_input)
```

함수

반환하지 않는 함수

- 반환하지 않는 함수란?

- 함수를 호출한 곳에 아무 결과값 전달 없이 문장만 수행
- 함수내 문장을 수행한 결과를 호출한 곳에서 필요없을 때 사용

```
alghost:3rd Alghost$ python3 011.py
Input: asdf
asdf
----------
Input: skip
Rejected
Input: quit
quit
----------
```

- 예제: 욕설

- 문자열을 입력
- (욕설은 자체

- return 사용 가능!

- 아무 값 없이 return 만 사용!
- 함수를 빠져나갈 때 사용가능

함수

반환하지 않는 함수

언제쓰일까..?

- 이메일 보내는 함수를 (가짜로..) 만들어보자!

```python
def send_mail(from_email, to_email, subject, contents):
    print("From:\t" + from_email)
    print("To:\t" + to_email)
    print("Subject: " + subject)
    print("Contents")
    print("*"*10)
    print(contents)
    print("*"*10)
    print("Completed")
    print("-"*10)

my_email = 'yskim@fastcampus.com'

users = []
users.append({'name':'john','email':'john@gmail.com'})
users.append({'name':'ray','email':'ray@gmail.com'})
users.append({'name':'jay','email':'jay@gmail.com'})

contents = 'Thank you'
for user in users:
    title = 'Dear. ' + user['name']
    send_mail(my_email, user['email'], title, contents)
```

함수

반환하는 함수

언제쓰일까

- 이메일 보내는

```
alghost:3rd Alghost$ python3 014.py
From:   yskim@fastcampus.com
To: john@gmail.com
Subject: Dear. john
Contents
**********
Thank you
**********
Completed
----------
From:   yskim@fastcampus.com
To: ray@gmail.com
Subject: Dear. ray
Contents
**********
Thank you
**********
Completed
----------
From:   yskim@fastcampus.com
To: jay@gmail.com
Subject: Dear. jay
Contents
**********
Thank you
**********
Completed
---------
```

함수

스코프

스코프란?

- Scope: 범위, 영역, 시야
- 변수는 처음 사용된 위치에 따라 스코프가 결정됨
- 스코프는 들여쓰기로 표현됨
- 함수에서의 스코프
  - 함수 안과 밖은 완전히 다른 공간!

```
def my_sum(a, b):
    result = a+b
    return result

print(result)
```

함수

스코프

스코프란?

- Scope: 범위, 영역, 시야
- 변수는 처음 사용된 위치에 따라 스코프가 결정됨
- 스코프는 들여쓰기로 표현됨
- 함수에서의 스코프
  - 함수 안과 밖은 완전히 다른 공간!

```
alghost:3rd Alghost$ python3 016.py
Traceback (most recent call last):
  File "016.py", line 5, in <module>
    print(result)
NameError: name 'result' is not defined
```

함수

스코프

스코프 예시

```
def send_mail(to_mail):
    from_email = 'alghost.lee@gmail.com'

    if not '@' in to_mail:
        other_mail = 'test@gmail.com'

    print('other: ' + other_mail)

send_mail('test')
send_mail('alghost@gmail.com')
```

```
alghost:3rd Alghost$ python3 018.py
other: test@gmail.com
Traceback (most recent call last):
  File "018.py", line 10, in <module>
    send_mail('alghost@gmail.com')
  File "018.py", line 7, in send_mail
    print('other: ' + other_mail)
UnboundLocalError: local variable 'other_mail' referenced before assignment
```

함수

자주쓰이는 내장함수 소개

내장함수

- 파이썬에서 기본적으로 제공하는 함수
- 역시나, 굉장히 많지만 자주 사용하는 함수에 대해 설명

내장함수	설명	예제
int	변수를 숫자형으로 변환하여 반환	num = int(<VAR>)
str	변수를 문자열로 변환하여 반환	string = str(<VAR>)
list	변수를 리스트로 변환하여 반환	list_var = list(<VAR>)
tuple	변수를 tuple로 변환하여 반환	tuple_var = tuple(<VAR>)

```
int_var = 'not number'
if int_var.isdigit():
    conv_var = int(int_var)
```

int()함수 사용시 문자열 내장함수 활용
isdigit(): 숫자로만 이뤄진 문자열인지 확인

함수

자주쓰이는 내장함수 소개

내장함수

내장함수	설명	예제
len	변수의 길이를 반환	length = len(<VAR>)
input	사용자로부터 문자열을 입력받음	user_input = input(‘’)
range	리스트를 만들어주는 함수	list_var = range(99)
max	리스트,튜플,문자열에서 최대값을 반환	max_var = max(<VAR>)
min	리스트,튜플,문자열에서 최소값을 반환	min_var = min(<VAR>)

클래스

왜 필요할까?

클래스란..?

- 굳이 얘기하자면, 프로그램을 위해 꼭! 필요한 요소는 아님
- 함수만으로는 코드 작성이 불편하여 편의를 위해 나옴
  - 예제를 통해서 확인

- **클래스란 변수와 함수의 집합!**
- 즉, 변수와 함수를 가진 나만의 템플릿
- 어떻게 생겼는지부터 보자

클래스

왜 필요할까?

클래스란..?

- 굳이
- 함수
- 클래
- 간단
  - 변
- 어떻

```
class 클래스이름():
    변수명 = 변수값

    def 함수명(self, 인자값 변수명, ...):
        실행할 코드
        실행할 코드
        실행할 코드

    def 함수명(self, 인자값 변수명, ...):
        실행할 코드
        실행할 코드
        실행할 코드
```

클래스

클래스의 기본

예제: 이메일 예제 활용!

- 기존: send_email 코드가 복잡하여 함수로 만듬
- email과 관련된 더 많은 기능을 구현하고 싶으면..?
  - email 보내기 위한 정보가 들어가있는지 확인하는 함수
  - email 주소가 올바른지 확인하는 함수
  - email 내용에 부적절한 단어가 들어있는지 확인하는 함수

클

클래스

예제

- 기존
- ema
  - e
  - e
  - e

```
def send_mail(...):
    print('send_mail')
    ...

def is_valid_email(...):
    print('is_valid_email')
    ...

def is_empty(...):
    print('is_empty')
    ...

def filter_bad(...):
    print('filter_bad')
    ...

from_email = 'alghost@gmail.com'
to_email = 'yskim@gmail.com'
subject = 'subject'
contents = 'contents'

if is_valid_email(from_email):
    print('valid')
if is_valid_email(to_email):
    print('valid')

if not is_empty(from_email, to_email, subject, contents):
    print('ok!')
```

이렇게 모든 기능을 함수로 만들고.. 데이터를 전달하면 되긴 된다!

클래스

클래스의 기본

예제: 이메

- 기존: send_
- email과 관
  - email 보
  - email 주소
  - email 내용

```
class Email():
    from_email = ''
    to_email = ''
    subject = ''
    contents = ''

    def is_valid_email(self):
        if '@' in self.from_email and '@' in self.to_email:
            return True
        else:
            return False

    def send_mail(self):
        print('send!')

    def filter_bad(self):
        if 'bad' in self.contents:
            print('bad!')


email = Email()
email.from_email = 'alghost.lee@gmail.com'
email.to_email = 'to@gmail.com'
email.subject = 'Dear. to'
email.contents = 'Hello'

if email.is_valid_email():
    email.send_mail()
```

이렇게 클래스 안에 변수와 함수를
구현해놓으면 “.”을 이용해 활용가능

클래스

클래스의 기본

하나하나 뜯어보자

- 클래스도 함수와 마찬가지로 호출되기전에는 수행 안됨!
- 클래스에 변수를 추가하고 사용해보자

클래스

클래스의 기본

하나하나 뜯어보자

- 클래스도 함수와 마찬가지로 호출되기전에는 수행 안됨!
- 클래스에 변수를 추가하고 사용해보자

```
class SimpleTest():
    a = 0

simple1 = SimpleTest()
simple2 = SimpleTest()
simple3 = SimpleTest()
print(simple1.a)
print(simple2.a)
print(simple3.a)

simple1.a = 10
simple2.a = 20
simple3.a = 30
print(simple1.a)
print(simple2.a)
print(simple3.a)
```

```
alghost:3rd Alghost$ python3 027.py
0
0
0
10
20
30
```

클래스

클래스의 기본

하나하나 뜯어보자

- 클래스에 함수를 만들어보자
- 함수에 써있는 self는?
  - 클래스의 변수에 접근하기 위해 파이썬이 제공하는 변수
  - 꼭, 잊지말고 써주어야 함

클래스

클래스의 기본

하나하나

- 클래스에 힘
- 함수에 써있
  - 클래스의 스
- 꼭, 잊지말

```
class SimpleTest():
    a = 0

    def print_without_skip(self, string):
        if 'skip' in string:
            print('Rejected')
            return
        else:
            print(string)

        print('-'*10)

simple = SimpleTest()

simple.print_without_skip('bad')
simple.print_without_skip('skip text')
simple.print_without_skip('alghost')
```

```
alghost:3rd Alghost$ python3 028.py
bad
----------
Rejected
alghost
----------
```

클래스

클래스의 기본

하나하나 뜯어보자

- 클래스에 함수를 만들어보자 2!
- 클래스의 함수에서 클래스의 데이터를 사용해보자
  - 앞에서 설명한 self를 사용

클래스

클래스의 기본

- 하나하나

- 클래스에

- 클래스의

- 앞에서

```
class SimpleTest():
    prefix = 'You said: '
    postfix = '\n'+'-'*20+'\n'

    def print_with_fix(self, string):
        print(self.prefix+string+self.postfix)

simple = SimpleTest()

simple.print_with_fix('bad')
simple.print_with_fix('skip text')
simple.print_with_fix('alghost')
```

```
alghost:3rd Alghost$ python3 031.py
You said: bad
--------------------

You said: skip text
--------------------

You said: alghost
--------------------
```

클래스

클래스의 기본

생성자란?

- 클래스 변수가 생성될 때 자동으로 호출되는 함수
- 클래스 내부에 정의된 변수 등을 초기화 할 때 사용

```
class SimpleTest():
    my_data = 0

    def __init__(self):
        self.my_data = 100
        print('Call init!')


simple = SimpleTest()
print(simple.my_data)
```

```
alghost:3rd Alghost$ python3 038.py
Call init!
100
```

클래스

클래스의 활용

개발할 기능이 여러개라면?

- 지금 배운대로 라면..
  - 한 파일에 계속 계속.. 계속.. 개발을.. => 알아보기도 힘들다
- 클래스를 이용해서 파일을 여러개로 나눠서 개발할 수 있다

자동화를 다 배운후에..

- 키워드를 포함한 뉴스 검색후에 엑셀에 저장하고 이메일 보내기!
  - auto.py 안에 뉴스 검색, 엑셀 저장, 이메일 보내기 구현해야함
  - 클래스를 활용한다면?
    - my_news.py => 뉴스, my_excel.py => 엑셀, my_email.py => 이메일, auto.py => 앞에 py들에 있는 클래스를 이용하여 자동화!

클래스

클래스의 활용

다른 파일의 클래스 사용

```
from <py파일 경로> import <함수 혹은 클래스명>
```

예제

```
from my_email import Email
from my_news import News
from my_excel import Excel

m_email = Email()
m_news = News()
m_excel = Excel()
```

클래스

auto.py

```
from my_email import Email
from my_news import News
from my_excel import Excel

m_email = Email()
m_news = News()
m_excel = Excel()

news_list = m_news.find_news('fastcampus')

m_email.from_email = 'alghost.lee@gmail.com'
m_email.to_email = 'yskim@fastcampus.com'
m_email.subject = 'Dear. '

for news in news_list:
    m_email.contents = m_email.contents + news + '\n'

m_email.send_mail()

m_excel.excel_file = 'result.xlsx'
m_excel.save_to_excel(news_list)
```

클래스

email.py

```
class Email():
    from_email = ''
    to_email   = ''
    subject    = ''
    contents   = ''

    def send_mail(self):
        print('Send to ' + self.to_email)
```

클래스

news.py


```
class News():
    news = []

    def find_news(self, keyword):
        for i in range(10):
            self.news.append('\''+str(i)+' news: '+keyword+'\'')
        print('completed')
        return self.news
```

클래스

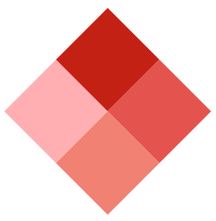
excel.py

```
class Excel():
    excel_file = ''
    def save_to_excel(self, list_data):
        for data in list_date:
            print('save ' + data + ' to ' + self.excel_file)
```

클래스

클래스가 진짜 필요한 이유

- 클래스의 집합 => 모듈, 라이브러리, 등…
- 전세계에 많은 개발자가 편리한 라이브러리를 만들어놓음
  - 프로그래밍이 쉬워졌다고 하는 "진짜 이유..!”
- 클래스를 만드는 방법에 익숙해지지 않아도 괜찮음!
  - 앞서 언급했듯이, 없어도 충분히 개발이 가능
- **하지만, 클래스내 변수와 함수를 “사용”하는 방법엔 익숙해져야함!**



Good Bye

See you next time

Appendix

파이썬 파일 폴더로 관리하기

from ~ import 심화

- from 하위는 경로가 될 수 있음 => 즉, 폴더/파일 가능
- 폴더의 경우 "."으로 표시할 수 있음
  - from <폴더명>.<폴더명>.<파일명> import <클래스명>

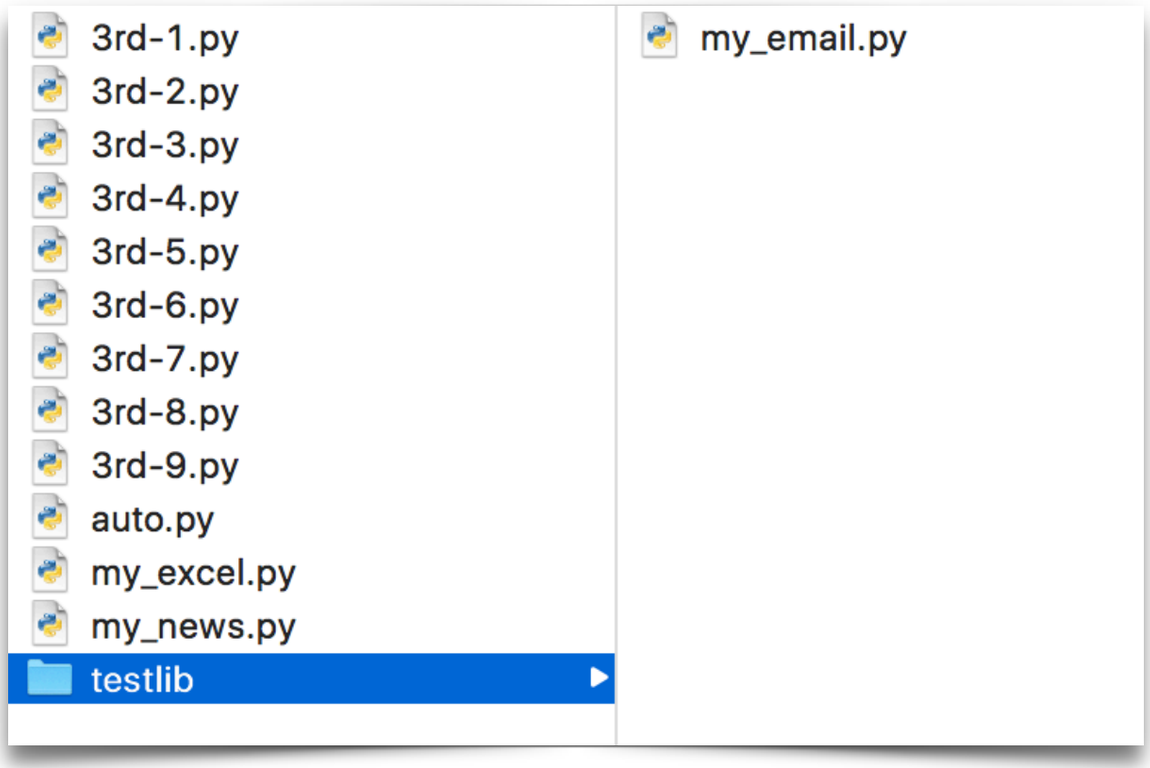
Appendix

파이썬 파일 폴더로 관리하기

from ~ import 심화

- my_email.py가 testlib 폴더 밑에 있을 때 사용하고 싶은 경우

```
from testlib.my_email import Email
from my_news import News
from my_excel import Excel
```



Appendix

클래스 파일 테스트하기

__name__ 내장변수 활용

- 파이썬에는 기본적으로 여러 내장변수, 내장함수가 있음
- 그 중, __name__은 해당 클래스를 어디에서 실행했는지 나타냄
- 아래 두 파일이 있고 import했을때와 실행했을때 차이를 확인

alghost.py

```
class Alghost():
    data = 100

    def print_data(self):
        print(self.data)

print(__name__)
```

appendix_01.py

```
from alghost import Alghost

a = Alghost()
a.print_data()
```

```
alghost:3rd Alghost$ python3 alghost.py
__main__
alghost:3rd Alghost$ python3 appendix_01.py
alghost
100
```

- from~import로 가져와도 안에 코드가 실행됨
- 클래스가 있는 파일 실행: __main__
- from~import로 실행: python 파일명

Appendix

클래스 파일 테스트하기

__name__ 내장변수 활용

- __name__을 활용하여 클래스 파일을 테스트할 수 있음
- 아래와 같이 테스트 코드를 작성하면 직접 실행할때만 실행됨
- from~import에 의해서는 실행이 안됨!

```
class Alghost():
    data = 100

    def print_data(self):
        print(self.data)

if __name__ == '__main__':
    test = Alghost()
    test.print_data()
```

Appendix

라이브러리 설치를 위한 확인

라이브러리 설치가 가능한 환경인지 확인

- Mac: 터미널 실행 후 “pip3” 실행 확인

```
alghost:3rd Alghost$ pip3

Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible dependencies.
  search                      Search PyPI for packages.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completion.
  help                        Show help for commands.

General Options:
  -h, --help                  Show help.
  --isolated                  Run pip in an isolated mode, ignoring environment variables and user configuration.
  -v, --verbose               Give more output. Option is additive, and can be used up to 3 times.
  -V, --version               Show version and exit.
  -q, --quiet                 Give less output. Option is additive, and can be used up to 3 times (corresponding to WARNING, ERROR, and CRITICAL logging levels).
  --log <path>                Path to a verbose appending log.
  --proxy <proxy>             Specify a proxy in the form [user:passwd@]proxy.server:port.
  --retries <retries>         Maximum number of retries each connection should attempt (default 5 times).
  --timeout <sec>             Set the socket timeout (default 15 seconds).
  --exists-action <action>    Default action when a path already exists: (s)witch, (i)gnore, (w)ipe, (b)ackup, (a)bort.
  --trusted-host <hostname>   Mark this host as trusted, even though it does not have valid or any HTTPS.
  --cert <path>               Path to alternate CA bundle.
  --client-cert <path>        Path to SSL client certificate, a single file containing the private key and the certificate in PEM format.
  --cache-dir <dir>           Store the cache data in <dir>.
  --no-cache-dir              Disable the cache.
  --disable-pip-version-check
                              Don't periodically check PyPI to determine whether a new version of pip is available for download. Implied with --no-index.
```

Appendix

라이브러리 설치를 위한 확인

라이브러리 설치가 가능한 환경인지 확인

- Windows: cmd 실행 후

  “pip” 실행 확인