PNU Mini Bootcamp 백엔드&클라우드 과정

1일차 - Python 기초 및 FastAPI 개요

송준우

2025년 2월 3일

멋쟁이사자처럼

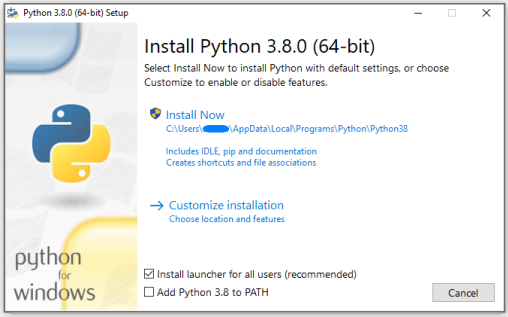
목차 i

1. Python 기초

1.1 Python 설치 - Windows

- https://www.python.org/downloads/windows/ 접속
- 시스템에 맞는 버전의 설치파일 다운로드하여 설치 진행
- 본 과정에서는 3.12 이상의 Stable Release를 추천합니다.

주의사항!



설치화면 하단의 **Add Python 3.* to PATH** 체크박스를 체크해주세요.

1.1 Python 설치 - 데비안 계열 Linux

대부분의 리눅스 배포판에는 Python이 기본적으로 설치되어 있습니다. 하지만, 최신 버전의 Python을 사용하고 싶다면 아래와 같이 설치할 수 있습니다.

터미널에서 아래 명령어를 입력하여 설치된 버전을 확인합니다.

```
1 $ python3 --version
```

설치된 Python 버전이 3.12 미만이라면 아래 명령어를 입력하여 Python 3.12버전을 설치합니다.

```
1 $ sudo apt install software-properties-common -y
2 $ sudo add-apt-repository ppa:deadsnakes/ppa
3 $ sudo apt-get update
4 $ sudo apt-get install python3.12
```

1.1 Python 설치 - 데비안 계열 Linux

데비안 계열 배포판에서 APT 패키지 매니저로 설치한 경우 Python 패키지 관리자가 기본적으로 설치되어 있지 않습니다. 아래 명령어를 입력하여 Python 패키지 관리자를 설치합니다.

터미널에서 아래 명령어를 입력하여 설치된 버전을 확인합니다.

```
$ pip3 -V
pip 22.X from ...site-packages/pip (python 3.12)
```

설치된 pip 버전이 Python 3.12에 호환되지 않는다면 아래 명령어를 입력하여 Python 3.12에 호환되는 버전으로 업그레이드합니다.

```
$ curl -sS https://bootstrap.pypa.io/get-pip.py | python3.12
```

1.1 Python 설치 - macOS

Python 사이트에서 제공하는 설치 파일 사용

- https://www.python.org/downloads/macos/ 접속
- 시스템에 맞는 버전의 설치파일 다운로드하여 설치 진행
- 본 과정에서는 3.12 이상의 Stable Release를 추천합니다.

Homebrew를 사용한 설치

터미널에서 아래 명령어를 입력하여 설치된 버전을 확인합니다.

```
$ python3 --version
```

설치된 Python 버전이 3.12 미만이라면 아래 명령어를 입력하여 Python 3.12버전을 설치합니다.

```
$ brew install python@3.12
```

1.2 IDE 설치 - Visual Studio Code

설치하기

- https://code.visualstudio.com/ 에서 시스템에 맞는 버전의 설치파일 다운로드하여 설치 진행

유용한 확장 프로그램

- Python
- Pylance
- Python Debugger
- Material Icon Theme
- indent-rainbow
- IntelliCode 등

1.3 개발문서 찾아보기

유용한 사이트

- https://docs.python.org/3
- https://developer.mozilla.org/
- https://fastapi.tiangolo.com
- https://stackoverflow.com
- https://www.reddit.com
- https://www.w3schools.com/python/default.asp

주의해야할 게시물

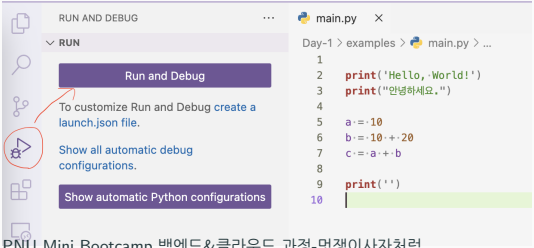
- Deprecated된 기능이나 버전이 포함된 게시물
- 더 이상 관리되지 않는 라이브러리나 프레임워크를 사용하는 게시물
- 깊은 이해 없이 복사 붙여넣기한 게시물 →키워드만 참고하고 공식 문서 찾아보기

1.4 기본문법 학습 (1) 디버거 실행

VSCode에서 Python 파일을 생성하고 아래 코드를 입력해보세요.

```
1 print("Hello, World!")
2 a = 10
3 b = 10 + 20
4 c = a + b
5 print('')
```

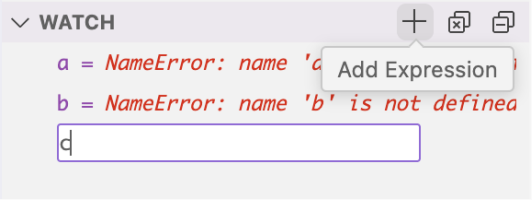
아래 그림처럼 디버그 모드로 실행해보세요.



1.4 기본문법 학습 (2) Watch

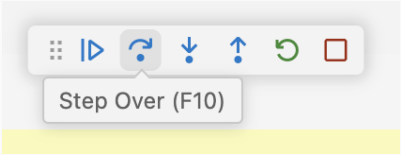
1. 아래 그림처럼 2번째 줄에 Breakpoint를 설정하고 디버그 모드로 실행해보세요.

2. 디버거가 시작되면 왼쪽 WATCH 창에서 아래와 같이 a,b,c를 입력해보세요.

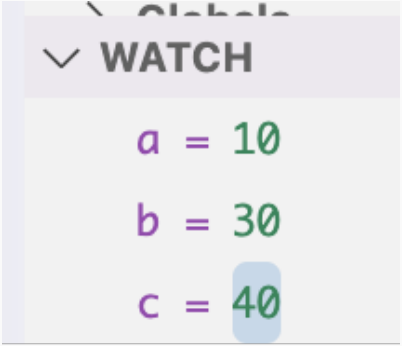


1.4 기본문법 학습 (2) Watch

3. Step Over 버튼을 클릭하거나 F10 키를 눌러보세요.



4. 코드를 한줄씩 실행하며 WATCH에 등록된 값을 확인해보세요.



5. 코드가 한줄씩 실행될때마다 메모리에 저장되는 값을 확인할 수 있습니다.

1.4 기본문법 학습 (3) 예외처리

VSCode에서 Python 파일을 생성하고 아래 코드를 입력 후 실행시켜보세요.

```
db = {
    '전자제품': [
        'MacBook', 'iPad', 'TV'
    ],
    '스포츠용품': [
        '축구공', '농구공', '야구공', '배구공'
    ]
}
strIn = '전자제품'
arRet = db[strIn]
print(arRet)
```

1.4 기본문법 학습 (3) 예외처리

dict에 없는 key로 접근하면 KeyError가 발생합니다.

```
strIn = '주방용품'
arRet = db[strIn] # KeyError 예외 발생
print(arRet)
```

이를 방지하기 위해서는 아래와 같은 두가지 대응 방법이 있습니다.

```
arRet = db.get(strIn, []) # 해당분류의 제품이 없으면 빈 리스트 반환
```

```
try: # try-except 구문을 사용하여 key가 없을 때 빈 배열 응답
    arRet = db[strIn]
except KeyError:
    arRet = []
```

1.4 기본문법 학습 (4) 함수와 자료형

아래와 같이 함수를 정의하고 호출해보세요.

```
db = {
    '전자제품': [
        'MacBook', 'iPad', 'TV'
    ],
    '스포츠용품': [
        '축구공', '농구공', '야구공', '배구공'
    ]
}
def get_products_of_category(category):
    return db.get(category, [])

electronics = get_products_of_category('전자제품')
products = get_products_of_category('주방용품')
print(electronics, products)
```

1.4 기본문법 학습 (4) 함수와 자료형

아래 주소로 접속하여 목업 데이터를 다운로드 받아주세요.
https://github.com/song9063/PNU-Bootcamp-2025/blob/main/Day1/products.py

footnotesize

```
    db = {
        'electronics': [
            "Apple iPhone 14",
            ...
        ],...
    }
```

Figure 1: products.py

다운로드 받은 파일을 활용하여 아래와 같이 코드를 작성해보세요.

```
from products import db

def get_products_of_category(category):
    return db.get(category, [])

electronics = get_products_of_category('electronics')
living = get_products_of_category('living')
print(electronics, living)
```

Figure 2: Day1-1.4.4.main.py

만약 아래와 같은 오류가 발생한다면

ImportError: attempted relative import with no known parent package

아래와 같이 소스코드가 있는 디렉토리에 __init__.py 파일을 생성해주세요.

```
Day1
├── __init__.py
├── products.py
└── main.py
```

Python 함수 구조

def 함수이름(입력값):
  함수내용
  return 반환값

- 입력값: 입력받을 값이 없다면 생략 가능
- 반환값: 반환할 값이 없다면 return문 생략 가능

```
def get_products_of_category(category):
    return db.get(category, [])
```

get_products_of_category 함수는 category 문자열을 입력받아 db에서 해당 카테고리의 상품 목록을 반환합니다.

1.4 기본문법 학습 (4) 함수와 자료형

쇼핑몰의 데이터베이스에는 수많은 상품이 있습니다. 이 상품들은 카테고리별로 분류되어 있습니다. 사용자가 카테고리를 선택했을때 해당 카테고리의 모든 상품 목록을 반환한다면 아래와 같은 문제점들이 발생합니다.

- API 응답을 만들기위해 DB의 모든 row를 스캔해야함
- DB의 모든 row를 메모리에 올려야함
- 클라이언트단에서 API응답 대기 시간이 길어짐
- 클라이언트단에서 응답을 받은 후 렌더링 시간이 길어짐
- 어차피 고객들은 전체 상품을 한번에 볼 수 없음

이러한 문제점을 해결하기 위해 Pagination을 사용합니다.

Pagination을 고려하여 get_products_of_category 함수 수정

- 입력 파라메터
  - 카테고리명: category
  - 페이지 번호(1부터 시작): page
  - 페이지당 상품 수: per_page
- 출력: 해당 카테고리의 상품 목록

위 요구사항을 고려하면 아래와 같이 함수를 수정할 수 있습니다.

```
def get_products_of_category(category, page, per_page):
```

Pagination을 고려한 코드

아래와 같이 get_products_of_category 함수의 본문을 수정하세요.

```python
from products import db
def get_products_of_category(category, page, per_page):
    startIndex = (page - 1) * per_page
    endIndex = startIndex + per_page
    items = db.get(category, [])
    return items[startIndex:endIndex]

electronics = get_products_of_category('electronics', 1, 5)
living = get_products_of_category('living', 1, 5)
print(electronics, living)
```

Figure 3: Day1-1.4.5.main.py

1.4 기본문법 학습 (4) 함수와 자료형

list: 다른 언어의 배열과 유사한 자료형으로써 아래와 같은 특징을 가집니다.

- list는 여러개의 값을 저장할 수 있는 자료형
- list는 []로 선언하며, 각 요소는 쉼표로 구분
- list는 인덱스로 요소에 접근 가능
- list는 슬라이싱을 통해 부분 리스트를 추출 가능
- list는 append, insert, remove, pop 등 다양한 메소드를 제공

list의 슬라이싱 기능을 사용하여 Pagination을 구현할 수 있습니다.

```
items[start:end:step]
```

list 슬라이싱 예시

슬라이싱을 사용하여 데이터 일부분 추출

```
nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(nums[0:5])   # [0, 1, 2, 3, 4]
print(nums[3:])    # [3, 4, 5, 6, 7, 8, 9]
print(nums[:4])    # [0, 1, 2, 3]
```

```
print(nums[-2:])   # [8, 9]
print(nums[-5:-2])   # [5, 6, 7]
```

```
print(nums[::2])   # [0, 2, 4, 6, 8]
print(nums[::-1])   # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

list 슬라이싱 예시

슬라이싱을 사용하여 데이터 일부분 변경

```
nums[2:5] = [20, 30, 40]
print(nums)  # [0, 1, 20, 30, 40, 5, 6, 7, 8, 9]
```

슬라이싱을 사용하여 데이터 일부분 삭제

```
nums[2:5] = []
print(nums)  # [0, 1, 5, 6, 7, 8, 9]
```

1.4 기본문법 학습 (4) 함수와 자료형

반복문을 사용하여 'electronics' 카테고리의 모든 상품을 출력해보세요.

```
page = 1
while True:
    electronics = get_products_of_category('electronics', page, 5)
    if len(electronics) == 0:
        break
    print(f'Page {page}: {electronics}')
    page += 1
```

Figure 4: Day1-1.4.6.main.py

문자열 앞에 f를 붙여 변수나 상수를 문자열에 삽입할 수 있습니다. [클릭: Python 공식문서]

함수 인자의 기본값 설정

get_products_of_category 함수의 per_page 값에 기본값을 설정하여 코드를 간략화할 수 있습니다.

아래와 같이 함수를 수정하여 예제코드를 실행해보세요.

```
def get_products_of_category(category, page, per_page=5):
    startIndex = (page - 1) * per_page
    endIndex = startIndex + per_page
    return db.get(category, [])[startIndex:endIndex]
```

```
    electronics = get_products_of_category('electronics', page)
```

1.4 기본문법 학습 (4) 함수와 자료형

get_products_of_category 함수의 page 인자에 숫자 1이 아닌 문자 '1'을 넣어봅시다.

```
from products import db

def get_products_of_category(category, page, per_page):
    startIndex = (page - 1) * per_page
    endIndex = startIndex + per_page
    return db.get(category, [])[startIndex:endIndex]

electronics = get_products_of_category('electronics', '1')
print(electronics)
```

Figure 5: Day1-1.4.7.main.py

5 실행결과

```
Traceback (most recent call last):
    File ".../Day1-1.4.7.main.py", line 8, in <module>
    electronics = get_products_of_category('electronics',...
    File ".../Day1-1.4.7.main.py", line 4, in get_pr...
    startIndex = (page - 1) * per_page
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

page 인자에 문자열 '1'을 넣었을 때 TypeError가 발생합니다.

1. 조건문을 사용하여 page가 숫자가 아닌 경우 예외처리

```
from products import db
def get_products_of_category(category, page, per_page=1):
    # https://docs.python.org/3/library/functions.html#type
    if type(page) != int:
        return None
    startIndex = (page - 1) * per_page
    endIndex = startIndex + per_page
    return db.get(category, [])[startIndex:endIndex]

electronics = get_products_of_category('electronics', '1')
if electronics is None:
    print('Invalid parameter')
print(electronics)
```

2. 타입 캐스팅을 통해 page 인자를 정수로 변환

```
def get_products_of_category(category, page, per_page=1):
    try:
        page = int(page)
    except ValueError:
        return None

    startIndex = (page - 1) * per_page
    endIndex = startIndex + per_page
    return db.get(category, [])[startIndex:endIndex]
```

page 인자를 정수로 변환 시도 후 실패하면 None을 반환합니다.

3. 함수 인자에 타입 힌트를 추가하여 함수 호출 전에 예외 발생시키기

함수 인자에 타입 힌트를 추가하여 함수가 호출되기전에 예외를 발생시킬 수 있습니다.
만약 다른 동료와 협업할때 타입 힌트를 통해 함수의 사용법을 알려줄 수 있습니다.

```
from products import db
def get_products_of_category(category: str, page: int, per_page: int=5):
    startIndex = (page - 1) * per_page
    endIndex = startIndex + per_page
    return db.get(category, [])[startIndex:endIndex]

# Exception raised when the parameter type is invalid
electronics = get_products_of_category('electronics', '1')
```

dataclass를 사용하여 인자의 타입 만들기

dataclass를 사용하여 인자의 타입을 만들어 함수의 가독성을 높일 수 있습니다.

```
from dataclasses import dataclass
from products import db

@dataclass
class ProductFetchParams:
    category: str
    page: int
    per_page: int = 5

def get_products_of_category(params: ProductFetchParams):
    startIndex = (params.page - 1) * params.per_page
    endIndex = startIndex + params.per_page
    return db.get(params.category, [])[startIndex:endIndex]
```

인자를 순서대로 입력하여 사용하기

dataclass에 정의된 순서대로 인자를 입력하여 사용할 수 있습니다.

```
paramForElectronics = ProductFetchParams('electronics', 1)
electronics = get_products_of_category(paramForElectronics)
```

인자명을 지정하여 사용 인자명을 명시적으로 지정하는 경우 인자의 순서는 중요하지 않습니다.

```
paramForLiving = ProductFetchParams(
    page=1,
    category='living',
    per_page=10
)
livings = get_products_of_category(paramForLiving)
```

함수 반환값의 타입 지정하기

함수의 반환값에 타입 힌트를 추가하여 함수가 반환하는 값의 타입을 명시할 수 있습니다.

함수의 반환값이 list인 경우 반환값의 타입을 list로 지정합니다.

```
def get_products_of_category(params: ProductFetchParams) -> list:
    startIndex = (params.page - 1) * params.per_page
    endIndex = startIndex + params.per_page
    return db.get(params.category, [])[startIndex:endIndex]
```

함수 반환값에 None 타입 추가하기

앞서 잘못된 인자를 넣었을 때 None을 반환하는 함수 예제 코드의 경우 반환값에 None 타입을 추가할 수 있습니다.

이렇게 하여 이 함수를 사용하는 개발자는 반환값이 None일 수 있음을 알 수 있습니다.

```
def get_products_of_category(category, page, per_page=1) -> list | None:
    if type(page) != int:
        return None
    startIndex = (page - 1) * per_page
    endIndex = startIndex + per_page
    return db.get(category, [])[startIndex:endIndex]
```

typing 모듈의 Optional 사용하여 Nullable 반환값 정의하기

```
from products import db
from typing import Optional

def get_products_of_category(category, page, per_page=1) ->
    Optional[list]:
    if type(page) != int:
        return None
    startIndex = (page - 1) * per_page
    endIndex = startIndex + per_page
    return db.get(category, [])[startIndex:endIndex]
```

dataclass, Optional을 사용하여 함수 타입힌트 정의하기 - ay1-1.4.8.main.py

```python
from dataclasses import dataclass
from typing import Optional
from products import db
@dataclass
class ProductFetchParams:
    category: str
    page: int = 1
    per_page: int = 5

def get_products_of_category(params: ProductFetchParams) -> Optional[list]:
    startIndex = (params.page - 1) * params.per_page
    endIndex = startIndex + params.per_page
    return db.get(params.category, [])[startIndex:endIndex]
```

Figure 6: Day1-1.4.8.main.py

category값에는 어떠한 문자열도 허용됩니다.
만약 카테고리명이 바뀐다면 코드내의 모든 카테고리명을 수정해야합니다.
이러한 문제를 해결하기 위해 **Enum**을 사용할 수 있습니다.
Enum은 수정 불가능한 상수의 집합입니다.

https://docs.python.org/3/library/enum.html

```
from enum import Enum
class ProductCategory(Enum):
    ELECTRONICS = 'electronics'
    CLOTHING = 'clothing'
    HOME = 'home'
```

ProductCategory는 타입으로 사용될 수 있습니다.

```python
from dataclasses import dataclass
from enum import Enum
class ProductCategory(Enum):
    ELECTRONICS = 'electronics'
    CLOTHING = 'clothing'
    HOME = 'home'

@dataclass
class ProductFetchParams:
    category: ProductCategory # ProductCategory 타입으로 변경
    page: int = 1
    per_page: int = 5
```

Enum이 적용된 dataclass 사용 예시

```
from products import db
from dataclasses import dataclass
from typing import Optional
from enum import Enum

class ProductCategory(Enum):
    ELECTRONICS = 'electronics'
    CLOTHING = 'clothing'
    HOME = 'home'
@dataclass
class ProductFetchParams:
    category: ProductCategory
    page: int = 1
    per_page: int = 5

def get_products_of_category(params: ProductFetchParams) -> Optional[list]:
    startIndex = (params.page - 1) * params.per_page
    endIndex = startIndex + params.per_page
    return db.get(params.category, [])[startIndex:endIndex]

params = ProductFetchParams(category=ProductCategory.ELECTRONICS)
electronics = get_products_of_category(params)
print(electronics)
```

1.5 JSON

https://www.w3schools.com/js/js_json_datatypes.asp

Data types in JSON, Python:

JSON	Python	설명
object	dict	Key-Value
Array	list	순서가 보장되는 데이터 집합
String	str	문자열
Number(int)	int	정수형 숫자
Number(float)	float	실수형 숫자
Boolean	bool	true=True, false=False
Null	None	null=None

JSON 데이터 타입과 Python 데이터 타입 비교

JSON →Python

```
{
    "id": 1,
    "name": "iPhone",
    "on_sale": true,
    "price": 1000.0,
    "colors": ["black", "white"],
    "option": {
        "storage": 128,
        "charger": null
    }
}
```

```
{
    "id": 1,
    "name": "iPhone",
    "on_sale": True,
    "price": 1000.0,
    "colors": ["black", "white"],
    "option": {
        "charger": None
    }
}
```

1.5 JSON - JSON 문자열 파싱하기

```python
import json
strJsonData = '''{
    "id": 1,
    "name": "iPhone",
    "on_sale": true,
    "colors": ["black", "white"],
    "option": {
        "storage": 128,
        "charger": null
    }
}'''
jsonData = json.loads(strJsonData)
nId = jsonData.get('id')
strName = jsonData.get('name', '')
nStorage = jsonData.get('option',{}).get('storage', 0)
arColors = jsonData.get('colors', [])
if jsonData.get('on_sale', False):
    print(f'Product {nId} {strName}({nStorage}) is on sale')
if len(arColors) > 0:
    print('Available colors:')
    for color in arColors:
        print(f'\t- {color}')
```

Listing 1: Day1-1.5.1.main.py

1.5 JSON - JSON 데이터 생성하기

```
import json
apiResponse = {
    "id": 1,
    "name": "iPhone",
    "on_sale": True,
    "price": 1000.0,
    "colors": ["black", "white"],
    "option": {
        "storage": 128,
        "charger": None
    }
}

strJsonData = json.dumps(apiResponse)
print(strJsonData)

strJsonData = json.dumps(apiResponse, indent=4)
print(strJsonData)
```

Listing 2: Day1-1.5.2.main.py