Python관련 면접 Q&A

`파이썬에서 리스트와 튜플의 차이점은 무엇인가요?`

리스트는 mutable(변경 가능), 튜플은 immutable(변경 불가능)이라는 특징을 가지고 있다. 따라서 리스트는 선언 후에도 값에 대한 변경, 삭제가 가능하지만, 튜플은 선언 후에 값을 변경하거나 삭제하는 것이 불가능하다. 또한 리스트는 튜플보다 느리다는 단점을 가지고 있으며, 하나의 튜플/리스트에 다른 타입의 값을 함께 저장할 수 있다는 공통점이 있다. 리스트는 대괄호 [ ]를, 튜플은 소괄호 ( )를 사용해서 나타낸다.

`파이썬의 주요 기능은 무엇인가요?`

파이썬이 주요 특징은 아래와 같다.

- **인터프리터 언어(Interpreter Language)**
  - 파이썬은 인터프리터 언어이므로, 실행하기 전에 컴파일을 할 필요가 없다.
- **동적타이핑(Dynamic Typing)**
  - 파이썬은 실행시간에 자료형을 검사하므로, 선언할 때 변수 유형(ex.int, double, ...)을 명시할 필요가 없다.
  - typing이란 프로그램 내에서 변수의 데이터 타입을 정하는 것을 말한다. 데이터 타입 지정(assign)은 정적 또는 동적 타이핑으로 분류되는데, 프로그램 컴파일 시에 변수의 타입을 체크하는 C, C++과 같은 언어는 정적 타입(static typed) 언어라고 하고, 프로그램 실행 시에 타입을 체크하는 python은 동적 타입(dynamic typed) 언어이다.
- **객체 지향 프로그래밍(OOP)**
  - 파이썬은 클래스와 구성 및 상속을 함께 정의할 수 있다는 점에서 객체 지향 프로그래밍에 매우 적합하다.
- **일급객체(First-class citizen)**
  - 파이썬에서 함수와 클래스는 일급 객체이다. 일급객체는 변수나 데이터 구조 안에 담을 수 있고, 매개변수로 전달이 가능하며, 리턴값으로 사용될 수 있다는 특징을 가지고 있다.

> 이 외 특징

- 파이썬은 **들여쓰기(indentation)** 와 간결하고 쉬운 문법을 통해 빠르게 코드를 작성할 수 있다는 장점을 가지고있다.

- 변수, 인수(argument)를 미리 선언하지 않아도 **자동으로 메모리 공간 할당**되어 편리하다.
- 함수(function) 또는 **모듈**(module) 추가가 용이하여 **확장성과 이식성**이 좋다.
- 파이썬은 인터프리터로 동작하는 **스크립트 언어**이므로 다른 컴파일 언어에 비해 다소 느리다.
    - 컴파일러가 코드를 기계어로 번역해서 실행가능 파일을 만드는 것에 비해, 인터프리터는 코드를 한줄씩 실행시간마다 번역해서 실행하기 때문이다.

`파이썬은 어떤 유형의 언어인가요? 프로그래밍인가요, 스크립팅인가요?`

파이썬은 정확하게는, 스크립트 언어이다. 모든 스크립트 언어는 프로그래밍 언어로 볼 수 있으나, 모든 프로그래밍 언어가 스크립트 언어로 분류되는 것은 아니다. 따라서 파이썬은 스크립트 언어이자, 프로그래밍 언어이다. 그러나 사람들은 일반적인 경우에 파이썬을 프로그래밍 언어의 목적으로 분류하고, 프로그래밍 목적으로 많이 사용한다.

> 💡 스크립팅(scripting/Scripting Language)
> 스크립트 언어란 컴파일이 필요없이 실행될 수 있는 명령어의 집합이다. 스크립트 언어는 인터프리터를 사용하는데, 인터프리터는 컴파일 과정이 필요하지 않으며, 소스코드로 부터 바로 명령어를 해석할 수 있다.

`파이썬은 해석된 언어입니다. 더 설명해 주시겠어요?`

인터프리터는 고급 언어로 작성된 원시코드 명령어들을 한번에 한 줄씩 읽어들여서 실행하는 프로그램이다. 인터프리터 언어는 실행시간(runtime) 전에 기계 레벨 코드(machine-level code)를 만드는 컴파일 언어와 다르게 소스코드를 바로 실행하는 언어이며, 파이썬은 인터프리터 언어에 해당한다.

인터프리터 언어는 스크립트 언어와 동일한 의미이다.

`PEP 8이란 무엇인가요?`

PEP(Python Enhancement Proposal)는 Python 코드를 포맷하는 방법을 지정하는 규칙 집합이다. 다른 사람과 원활하게 협업하려면 공통된 스타일 공유가 필요하며, 일관성 있는 스타일은 나중에 수정하기도 쉽다. PEP8은 파이썬 코드를 어떻게 구성할 지 알려주는 스타일 가이드로서의 역할을 한다. Python formatting tool에는 black, flake8, autopep8, yamf 등이 있다.

PEP8 스타일 가이드 예시

- whitespace
  - 한 줄의 문자 길이가 79자 이하여야 한다.
  - 함수와 클래스는 빈 줄 두개로 구분한다.
- naming
  - 함수, 변수, 속성 : lowercase_underscore
  - 보호(protected) 인스턴스 속성 : _leading_underscore
  - 비공개(private) 인스턴스 속성 : __double_leading_underscore

`파이썬에서 메모리는 어떻게 관리되나요?`

Python은 모든 것을 객체로 관리한다. 객체가 더이상 필요하지 않으면 파이썬 메모리 관리자가 자동으로 객체에서 메모리를 회수하는 방식을 사용하므로, 파이썬은 **동적 메모리 할당** 방식을 사용한다고 말할 수 있다. **힙(heap)**은 동적할당을 구현하는데 사용된다. 힙을 사용하여 동적으로 메모리를 관리하면, 필요하지 않은 메모리를 비우고 재사용할 수 있다는 장점이 있다. 모든 파이썬 객체 또는 자료구조는 **python private heap** 공간에서 관리되며, 프로그래머는 이 공간에 접근할 수 없고, 대신 파이썬 인터프리터가 대신해서 관리한다.

파이썬 객체에 대한 힙 공간 할당을 담당하는 것을 **파이썬 메모리 관리자(Python Memory Manager)** 라고 부른다. Python 메모리 관리자에는 객체별 할당자가있기 때문에 int, string 등과 같은 특정 객체에 대해 메모리를 명확하게 할당 할 수 있다. 또한, 사용되지 않는 모든 메모리를 재활용하고 힙 공간에서 사용할 수 있도록 하는 **내장 Garbage Collector(GC)** 를 가지고 있다.

`파이썬에서 네임스페이스란 무엇인가요?`

namespace는 이름 지정 충돌(naming conflicts)을 피하기 위해 이름이 고유한지 확인하는 데 사용되는 이름 지정 시스템(naming system)이다. 네임스페이스(namespace, 이름 공간)란 프로그래밍 언어에서 특정한 객체(Object)를 이름(Name)에 따라 구분할 수 있는 범위를 의미한다. 파이썬 내부의 모든것은 객체로 구성되며 이들 각각은 특정 이름과의 매핑 관계를 갖게 되는데 이 매핑을 포함하고 있는 공간을 네임스페이스라고 한다.

네임스페이스가 필요한 이유는 다음과 같다. 프로그래밍을 수행하다보면 모든 변수 이름과 함수 이름을 정하는 것이 중요한데 이들 모두를 겹치지 않게 정하는 것은 사실상 불가능하다. 따라서 프로그래밍언어에서는 네임스페이스라는 개념을 도입하여, 특정한 하나의 이름이 통용될 수 있는 범위를 제한한다. 즉, **소속된 네임스페이스가 다르다면 같은 이름이 다른 개체를 가리키도록 하는 것이 가능**하다.

파이썬 네임스페이스의 특징

- 네임스페이스는 딕셔너리 형태로 구현된다.
- 모든 이름 자체는 문자열로 되어있고 각각은 해당 네임스페이스의 범위에서 실제 객체를 가리킨다.
- 이름과 실제 객체 사이의 매핑은 가변적(Mutable)이므로 런타임동안 새로운 이름이 추가될 수 있다.
- 빌트인 네임스페이스는 함부로 추가하거나 삭제할 수 없다.

파이썬 네임스페이스의 3가지 분류

- **빌트인 네임스페이스(build-in namespace):** 기본 내장 함수 및 기본 예외들의 이름들이 소속된다. 파이썬으로 작성된 모든 코드 범위가 포함된다.
- **전역 네임스페이스(global namespace):** 모듈별로 존재하며, 모듈 전체에서 통용될 수 있는 이름들이 소속된다.
- **지역 네임스페이스(local namespace):** 함수 및 메서드 별로 존재하며, 함수 내의 지역 변수들의 이름들이 소속된다.

`파이썬 경로란 무엇인가요?`

모듈을 import할 때 사용되는 환경변수이다. 모듈을 import할 때마다 PYTONPATH를 조회하여 가져온 모듈이 디렉토리에 있는지 확인한다. 인터프리터는 이를 사용하여 로드할 모듈을 결정한다.

PYTHONPATH 환경 변수에 경로를 추가하면, 파이썬은 이 경로들을 sys.path에 추가한다. 이를 통해 파이썬 코드 내부에서 뿐만 아니라 파이썬 코드 밖에서도 sys.path를 조작할 수 있다. PYTHONPATH에는 sys.path에 추가할 여러 경로들이 들어간다. 리눅스에서는 /foo:/bar 처럼 : 로 두 경로를 구분하고, 윈도우에서는 /foo;/bar 처럼 ; 로 두 경로를 구분한다. 이외에도 sys.path에는 파이썬에 포함된 여러 내장 모듈 등을 탐색하기 위한 기본 경로가 들어간다.

sys.path의 순서

import는 sys.path 리스트에 들어있는 경로들을 탐색하며 불러올 파이썬 파일을 찾는다. 리스트에 들어있는 맨 처음 경로부터 탐색을 시작하여, 특정 경로에서 불러올 파일을 찾았다면 남은 경로를 더 찾아보지 않고 탐색을 중지한다. sys.path의 기본값은 아래의 순서대로 추가된다.

- .py 파일이 속한 디렉터리의 절대 경로
- PYTHONPATH 환경 변수
- 기타 기본 경로

아래의 코드를 통해서 sys.path를 직접 확인할 수 있다.

Copy

```
import sys
print(sys.path)
```

`파이썬 모듈이란 무엇인가요? 파이썬에서 일반적으로 사용되는 내장 모듈의 이름은 무엇인가요?`

모듈이란 Python 코드를 포함하는 파일로써, 함수나 변수 또는 클래스를 모아 놓은 파일이다. 모듈은 다른 파이썬 프로그램에서 불러와 사용할 수 있게끔 만든 파이썬 파일이라고도 할 수 있다. 실행 가능한 코드를 포함하는, 파이썬 확장자 .py로 만든 파이썬 파일은 모두 모듈이다. 모듈을 사용하면, 다른 코드에 적용하기가 쉬워지므로 이식성이 좋아진다.

자주 사용되는 빌트인 모듈(built-in module)의 예시는 다음과 같다.

- os
- sys
- math
- random
- datetime
- JSON

`파이썬에서 지역 변수와 전역 변수는 무엇인가요?`

- *전역 변수(Global Variable)**는 함수 외부 또는 전역 공간에 선언된 변수이다. 프로그램의 모든 함수에서 전역변수에 접근할 수 있다.
- *로컬 변수(Local Variable)**는 함수 내부에 선언된 변수를 말한다. 로컬 변수는 전역 공간이 아닌 로컬 공간에 있다.

Copy

```
a=2
def add():
    b=3
    c=a+b
    print(c)
add()

# 출력: 5

# global var: a
# local var: b, c
```

add()함수의 외부에서 add() 함수의 로컬 변수에 액세스하려고 하면 오류가 발생한다.

`파이썬은 대소문자를 구분하나요?`

파이썬은 대소문자를 구분하는 언어이다. 예를들어, a와 A는 다른 변수이다.

`파이썬에서 타입 변환이란 무엇인가요?`

type conversion은 타입 캐스팅(type casting)과 동일한 의미를 가지며, 이는 어떤 데이터 타입을 다른 데이터 타입으로 변환하는 것을 말한다.

> 타입 캐스팅 함수의 종류

- **int():** 정수형으로 변환한다.
- **float():** 실수형으로 변환한다.
- **ord():** 문자형을 정수형으로 변환한다.
- **hex():** 정수형을 10진수로 변환한다.
- **oct():** 정수형을 8진수로 변환한다.
- **tuple():** 튜플형으로 변환한다.
- **set():** set으로 변환한다.
- **list():** list로 변환한다.
- **dict():** (key,value) 순서로 이뤄진 튜플을 딕셔너리형으로 변환한다.
- **str():** 정수형을 문자형으로 변환한다.

- **complex(real, image):** 실수를 복소수로 변환한다.

`윈도우에 파이썬을 설치하고 경로 변수를 설정하는 방법은 무엇인가요?`

Windows에 Python을 설치하려면 다음 단계를 거쳐야한다.

1. 파이썬 홈페이지에서 python을 설치한다.
2. PC에 다운로드 받은 python을 설치하면서, Add Python 3.6 to PATH에 체크하고, 안내에 따라 설치하며 python을 설치한 위치를 저장해둔다.
3. 시스템 > 시스템 정보 > 고급 시스템 설정 > 환경변수로 이동하여 시스템 변수를 편집하여 2번에서 저장해둔 python.exe 실행파일이 있는 경로를 추가해주면 된다.

`파이썬에서 들여쓰기는 필수인가요?`

Python은 Indentation(들여쓰기)이 필요하다. 파이썬은 { }을 사용하여 영역을 지정하지 않고, 들여쓰기를 사용하여 코드블록을 지정하기 때문에 파이썬에서 들여쓰기는 문법적인 강제사항이다. if, for, class, def 등의 모든 코드는 들여쓰기 블록 내에서 지정된다. 들여쓰기의 방법은 1칸, 2칸, 4칸, 탭 등 여러가지 방식이 있다. 일반적으로 파이썬은 네 개의 공백 문자를 사용하여 들여쓰기를 수행한다.

코드가 정확하게 들여쓰여지지 않으면 실행되지 않고 오류도 발생한다. 중요한 것은 같은 블록 내에서는 들여쓰기 칸 수가 같아야한다는 것이다. 들여쓰기 규칙 위반시에는 IndentationError: unexpected indent 에러를 출력한다.

`파이썬 array과list의 차이점은 무엇인가요?`

list는 다양한 데이터 타입을 허용하며, array는 보통 숫자 데이터로 제한된다. 배열은 주로 numpy 라이브러리를 사용하여 생성한다. array의 선언 방법은 arrayName = array(type, [Values]) 처럼 자료형을 정하고, 지정한/동일한 자료형만을 넣을 수 있도록 되어있다. list은 변수에 [ ]로 여러 타입의 변수를 묶어서 선언할 수 있다.

Copy

```
import array as arr

My_Array=arr.array('i',[1,2,3,4])
My_list=[1,'abc',1.20]
print(My_Array)
print(My_list)
```

```
# Output: array('i', [1, 2, 3, 4]) [1, 'abc', 1.2]
```

`파이썬에서 함수는 무엇인가요?`

함수는 호출될 때만 실행되는 코드 블록이다. Python 함수를 정의하기 위해 def 키워드가 사용된다. 반복되는 부분을 함수로 만들어서 사용하면, 똑같은 코드를 여러번 반복하여 쓰지 않아도 되고, 프로그램의 흐름을 쉽게 파악할 수 있다는 장점이 있다.

Copy

```
def new_func():
  print("Hi, Welcome to Edureka")

new_func(); # 함수 호출

# Output: Hi, Welcome to Edureka
```

`__init__이란 무엇인가요?`

__init__는 파이썬에서 특별하게 약속된 메서드 가운데 하나로, 초기화 메서드 혹은 생성자라고도 한다. 이 메서드는 클래스의 새 개체/인스턴스가 생성될 때 메모리를 할당하기 위해 자동으로 호출되며, 그 객체가 갖게 될 여러 가지 성질을 정해준다. 모든 클래스에는 __init__메서드가 있다.

Copy

```
class Employee:
  def __init__(self, name, age,salary):
    self.name = name
    self.age = age
    self.salary = 20000

E1 = Employee("XYZ", 23, 20000)

# E1은 Employee 클래스의 객체
# __init__ 는 E1에 메모리를 할당함

print(E1.name)
```

```
print(E1.age)
print(E1.salary)

'''
출력:

XYZ
23
20000
'''
```

> 💡 __init__.py은 무엇인가?
> python 3.3 이하 버전에서, package import하기 위해서 사용되는 규칙이다. 3.3 이후의 버전에서는 이 제약사항이 해제되었다.
> __init__.py는 python 프로그램이 디렉토리를 처음 가져올 때 자동으로 실행되는 패키지 초기화 파일 역할을 하고, 모듈의 네임스페이스 초기화 역할을 한다.

`람다 함수란 무엇인가요?`

익명 함수(이름이 없는 함수)를 람다 함수라고 한다. 람다 함수는 def 키워드를 통해서 함수를 생성하는 리터럴 표기법을 **딱 한 줄의 코드로 표현**할 수 있게 해주며, lambda 인자 : 표현식의 형식으로 표현한다. 람다함수는 결과 부분을 return 키워드 없이 자동으로 return한다. 람다함수를 사용하면 코드가 간결해지고 메모리가 절약된다는 장점이 있다. 그러나 함수에 이름이 없고, 저장된 변수가 없기 때문에 다시 사용하기 위해서는 다시 코드를 적어주거나, 람다함수를 변수에 담아주어야한다. 따라서, 재사용할 이유가 없다면 lambda 함수를 생성하여 넘겨주는 편이 좋다.

람다함수의 표현법을 그림으로 표현하면 아래와 같다.

Copy

```
a = lambda x, y : x + y
print(a(5, 6))

# Output: 11
```

`파이썬에서 자식이란 무엇인가요?`

Copy

```
class MyClass:
    def method(self):
        return 'instance method', self

obj = MyClass
print(obj.method())

# >> ('instance method', <__main__.MyClass object at 0x7f10
aa8e68b0>)
```

우선 **self**가 어디에서 쓰이는지 알아야 한다. **self**는 인스턴스 메서드(instance method)의 첫 번째 인자이다. 메서드가 호출될 때, 파이썬은 **self**에 인스턴스를 넣고 이 인스턴스를 참조하여 인스턴스 메서드를 실행할 수 있게 된다.

`break, continue, pass work는 어떻게 작동하나요?`

break는 가장 가까운 for문이나 while문의 루프에서 빠져나가도록 한다.

Copy

```
for i in range(10):
  if i == 5:
    break
  print(i, end=' ')

# >> 0 1 2 3 4
```

continue는 이번 이터레이션(iteration)을 건너뛰고 다음 이터레이션을 이어나가도록 한다.

Copy

```
for i in range(10):
  if i == 5:
    continue
  print(i, end=' ')
```

```
# >> 0 1 2 3 4 6 7 8 9
```

pass는 문법적으로 필요하지만, 아무 것도 하지 않게 하고 싶을 때 사용한다. 주로 함수나 클래스의 구조부터 세우고 나중에 구현을 하고 싶을 때 사용한다.

Copy

```
class MyClass:
    def not_implemented_method(self):
        pass
```

`[::-1] 은 무엇을 하나요?`

파이썬 시퀀스 자료형은 값이 연속적으로 이어진 자료형으로, **리스트, 튜플, range, 문자열**이 있다. 시퀀스 자료형은 시퀀스 객체의 일부를 잘라낼 수 있는 **슬라이싱(slicing)**이라는 기능을 쓸 수 있다. 슬라이싱은 seq[start:end:step] 처럼 쓸 수 있으며, start는 시작 인덱스, end는 끝 인덱스(범위에 포함하지는 않음), step은 인덱스 증감폭을 말한다. step이 양수이면 증가하고, 음수이면 감소한다.

다시 돌아와 seq[::-1]은 start와 end는 시작 인덱스와 끝 인덱스를 생략하였는데, 이럴 경우 전체 시퀀스를 가져오며, 증감폭이 -1이므로 end-1부터 시작해 start순으로 요소를 가져온다. 즉, seq[::-1]은 시퀀스를 역전(reverse)시킨다.

`파이썬에서 목록의 항목을 어떻게 무작위화할 수 있나요?`

random 모듈의 shuffle 메서드를 사용하면 구현할 수 있다. random.shuffle은 시퀀스 객체의 요소를 임의로 섞어서 해당 시퀀스를 반환한다.

Copy

```
import random

random.seed(2021)       # 시드 고정
lst = list(range(10))
print(lst)              # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
random.shuffle(lst)
print(lst)              # [2, 7, 9, 3, 0, 5, 1, 4, 8, 6]
```

`이터레이터와 이터러블의 차이점은 무엇인가요?`

Copy

```
lst = [1, 2]            # iterable 객체 : 리스트
lst_iter = iter(lst)    # iterator 객체

print(next(lst_iter))   # 1
print(next(lst_iter))   # 2
print(next(lst_iter))   # StopIteration 예외 발생
```

iterable 객체는 iter 함수에 인자로 전달 가능한, 반복 가능한 객체를 말한다. 예를 들면, 리스트(list), 딕셔너리(dictionary), 집합(set), 문자열(string) 등이 있다.

iterable 객체를 iter 함수의 인자로 넣으면 iterable 객체를 순회할 수 있는 객체를 반환하는데, 이것이 iterator 객체이다. iterator 객체를 next 함수의 인자로 주면 iterable 객체의 요소의 값을 차례대로 반환한다. 만약 iterable 객체를 모두 순회했다면, *StopIteration* 예외를 발생시킨다. 만약 다시 순회를 하고 싶다면 iter 함수로 새로운 iterator 객체를 생성해 주면 된다.

`파이썬에서 난수를 생성하려면 어떻게 해야 하나요?`

random 모듈로 간단히 생성할 수 있다. random() 함수는 0과 1 사이의 부동소수점 값을 생성하며, randint()는 특정 범위 내의 정수를 생성한다. 편의를 위해 다음과 같이 random 모듈을 import하고 시드값을 2021로 고정하자.

Copy

```
import random
random.seed(2021)
```

- 0과 1사이의 임의의 수를 생성하고 싶은 경우

Copy

```
print(random.random())          # 0.8363375023320742
```

- 특정 범위 내의 임의의 정수를 생성하고 싶은 경우

Copy

```
print(random.randint(0, 10))    # 10
```

- 특정 범위 내의 n개의 정수를 생성하고 싶은 경우

Copy

```
n = 5
lst = range(1, 101)
print(random.sample(lst, 5))    # [70, 36, 32, 82, 5]
```

`range와 xrange의 차이점은 무엇인가요?`

> 파이썬2에서는 range와 xrange 모두 존재하지만, 파이썬3부터는 range가 내부적으로 xrange로 동작하도록 바뀌어서 range만 존재한다. 그러므로 파이썬2를 기준으로 range와 xrange를 설명한다.

range객체는 입력으로 받은 정수 범위의 값을 요소로 같는 리스트를 말한다. 그러므로 range(3)과 [0, 1, 2]는 완전히 동일하다.

Copy

```
# python2
r = range(5)
print(r)            # [0, 1, 2, 3, 4]
print(type(r))      # <type 'list'>
```

xrange는 제너레이터 객체로, 오직 루프를 돌때만 해당 범위의 정수를 하나씩 반환한다.

Copy

```
#python2
r = xrange(5)
print(r)            # xrange(5)
print(type(r))      # <type 'xrange'>

for i in r:
    print i,
# >> 0 1 2 3 4
```

`python에서 주석은 어떻게 작성하나요?`

#을 여러 줄 사용하여 여러 줄의 주석을 달 수 있다.

Copy

```
# this is my comment
# I am commenting multiple lines
# - boostcamp ai tech team 4
```

`pickling과 unpickling이 무엇인가요?`

우선 직렬화(Serialization)와 역 직렬화(Deserialization)의 개념을 알아야 한다. 직렬화란 객체를 바이트 스트림(byte stream)으로 변환하여 디스크에 저장하거나 네트워크로 보낼 수 있도록 만들어주는 것을 말한다. 반대로 바이트 스트림을 파이썬 객체로 변환하는 것을 역 직렬화라고 한다.

pickle 모듈은 파이썬 객체의 직렬화와 역 직렬화를 수행하는 모듈이다. 이 때 파이썬 객체를 직렬화할 때를 pickling이라고 하며, 바이트 스트림을 역 직렬화할 때를 unpickling이라고 한다.

`파이썬에서 제너레이터란 무엇인가요?`

제너레이터(Generator)란 Iterator 객체를 간단히 만들 수 있는 함수를 말한다. 제너레이터는 다음과 같이 1) yield문과 함수, 2) 표현식 형태로 만들 수 있다.

방법 1. yield문과 함수

- 제너레이터 함수 정의

Copy

```
def generator_list(value):
    for i in range(value):
        # 값을 반환하고 여기를 기억
        yield i
```

- 제너레이터 객체 생성 및 next 함수로 호출

Copy

```
gen = generator_list(2)
print(next(gen))    # 0
```

```
print(next(gen))    # 1
print(next(gen))    # StopIteration 에러 발생
```

방법 2. 표현문

Copy

```
value = 2
gen = (i for i in range(value))
print(next(gen))    # 0
print(next(gen))    # 1
print(next(gen))    # StopIteration 에러 발생
```

그럼 왜 리스트 대신 제너레이터를 사용할까? 리스트를 사용하면 리스트의 크기만큼 메모리에 공간이 할당된다. 반면 제너레이터는 말그대로 next 함수로 호출될 때 값을 생성하고 해당 값만 메모리에 올린다! 즉, 메모리를 절약할 수 있다. 작은 데이터라면 상관없지만 큰 데이터에서는 제너레이터 사용이 필수이다.

`문자열의 첫 글자는 어떻게 대문자로 표시하나요?`

문자열 메서드 capitalize를 사용하면 된다.

Copy

```
string = "boostcamp ai tech"
print(string.capitalize())      # Boostcamp ai tech
```

`문자열을 모두 소문자로 변환하려면 어떻게 하나요?`

문자열 메서드 lower을 사용하면 된다.

Copy

```
string = "BOOSTCAMP AI TECH"
print(string.lower())           # boostcamp ai tech
```

`Python에서 docstrings이란 무엇인가요?`

What are docstrings in Python?

docstrings은 주석은 아니지만, 사용자에게 코드에 대한 설명을 적어놓은 문서(documentation)이다. docstrings는 __doc__ 속성이나 help() 내장 함수로 접근할 수 있다. docstrings는 작은 따옴표(') 혹은 큰 따옴표(") 3개로 작성할 수 있다.

Copy

```
def mult(a, b):
  """
  Returns the product of a and b
  - a(float): any real number
  - b(float): any real number
  """
  return a*b
```

Copy

```
print(help(mult))
print(mult.__doc__)
```

> 💡 Comments(주석) vs Dosctrings
> comments와 docstrings은 각각 #, """을 쓴다는 점에서 다르지만 가장 큰 차이는 읽는 대상이다. comments는 개발을 위해 동료 혹은 나중에 코드를 읽을 나에게 남겨놓는 것이고 docstrings는 이 코드를 사용할 사용자들이 이해하기 쉽도록 남겨놓는 것이다.

`연산자에서 is, not 및 not의 용도는 무엇인가요?`

is는 객체 비교 연산자(identity operator)로 두 변수가 참조하는 객체의 id가 같을 경우 **True**를 반환한다. 보통 두 변수가 참조하는 객체가 동일한 객체인지 확인할 때 사용한다.

Copy

```
a = [1, 2, 3]
b = a
c = a.copy()
```

```
print(a is b) # True
print(a is c) # False
```

not은 단항 논리 연산자(logical operator)로 뒤에 오는 boolean 값을 뒤집는다. 뒤에 오는 값이 **True**이면 **False**를, **False**이면 **True**를 반환한다.

Copy

```
print(not True)   # False
print(not False)  # True
```

in은 멤버 연산자(membership operator)로, 요소 a와 시퀀스 b가 있는 지를 확인하고 싶을 때 a in b로 표현하며 만약 a가 b 안에 있다면 **True**를, 없으면 **False**를 반환한다.

Copy

```
b = "abc"
print("a" in b) # True
print("z" in b) # False
```

`Python에서 help() 및 dir() 함수의 용도는 무엇인가요?`

help()는 docstrings를 작성하였다면 해당 docstrings를 출력한다. docstrings에는 클래스, 메서드의 사용법에 관한 내용이 담겨있으므로 해당 클래스와 메서드를 사용자에게 매우 유용하다.

dir()은 인자로 넣은 객체의 속성과 메서드를 문자열로 변환하고 그것을 요소로 갖는 정렬된 리스트를 반환한다. dir은 사용할 객체의 메서드와 속성에 대한 정보를 얻고 싶을 때 유용하다. 다만 인자가 없다면 현재 지역 스코프에서 정의된 함수와 변수들의 리스트를 반환한다.

Copy

```
def func(x):
  return x

a = 3
print(dir(a))     # 객체 a에 대한 속성, 메서드
print(dir(func))  # 함수 func에 대한 속성, 메서드
print(dir())      # 지역 스코프에 정의된 a와 func
```

`Python이 종료될 때마다 모든 메모리가 할당 해제되지 않는 이유는 무엇인가요?`

다른 객체나 전역 네임스페이스에서 참조되는 객체를 순환 참조하는 파이썬 모듈은 항상 해제되지는 않는다. 또한 C 라이브러리가 예약한 메모리의 해당 부분을 해제하는 것은 불가능하다. 그러므로 파이썬 종료 시, 모든 메모리가 해제되지는 않는다.

> 💡 순환 참조(Circular Reference)
> 두 객체가 서로 참조하는 경우를 말한다.

> 💡 전역 네임스페이스(Global Namespace)
> 네임스페이스(namespace)란 프로그래밍 언어에서 특정 객체를 이름에 따라 구분할 수 있는 범위를 의미한다. 전역 네임스페이스는 import 한 모듈들의 이름을 포함하며, 스크립트가 끝날 때까지 지속된다.

`Python에서 dictionary는 무엇인가요?`

딕셔너리는 **key값과 그에 대응하는 value값을 얻을 수 있는 컬렉션**을 말한다. 딕셔너리는 데이터가 들어온 순서가 상관이 없고, 인덱싱이 되어 있어 key값으로 요소에 접근하여 데이터(= value) 수정이 가능하다. 하지만, key값은 고유 값이므로 key값 중복은 불가능하다. 주로 자체적으로 만든 key값으로 데이터에 접근하고 싶을 때 딕셔너리 컬렉션을 사용한다.

딕셔너리의 뜻은 사전이다. 영한 사전에서 각 영단어(ex. beautiful)에 대응하는 단어(ex. 아름다운)가 나오는 것처럼, 영단어가 key값이고 그에 대응하는 단어를 value값으로 볼 수 있다.

> 특징1 : 딕셔너리는 {, }를 사용하여 선언하며 { key1 : value1, key2 : value2, ... } 로 요소를 나타낸다.

- key값으로 변하지 않는 값을 사용하고, value값으로 변하는 값과 변하지 않는 값 둘 다 사용할 수 있다.
- key값으로 리스트를 사용하면, 값이 변할 가능성이 있기 때문에 인터프리터에서 type error를 발생시킨다.

Copy

```
ex1 = {'name':'Groot', 'lover':'penguin', 'feature':'really cute'}
```

```
# 딕셔너리 생성자로 집합을 생성하는 경우
ex2 = dict(name='Groot', lover='penguin', feature='really c
ute')

# key값은 변하지 않는 값, value값은 변하지 않는 값과 변하는 값 둘 다
가능
ex3 = {[10, 3]:'birthday'}  # type error!
```

특징2 : 딕셔너리는 딕셔너리 쌍(key : value)을 추가하거나 제거할 수 있다.

- 추가 : dict_ex[새로운 key값] = 그에 대응하는 value값으로 추가할 수 있다.
- 제거 : del 키워드를 이용해 특정 쌍을 제거할 수 있다.
  - 딕셔너리 앞에 del 키워드를 쓰면 딕셔너리가 완전히 제거된다.

Copy

```
ex = {'Kevin':180, 'Anna':165, 'Penelope':175}

# 새로운 딕셔너리 쌍 추가
ex['Groot'] = 100
print(ex)       # {'Kevin': 180, 'Anna': 165, 'Penelope': 1
75, 'Groot': 100}

# del 키워드로 딕셔너리 쌍 제거
del ex['Penelope']
print(ex)       # {'Kevin': 180, 'Anna': 165, 'Groot': 100}

# del 키워드로 딕셔너리 완전 제거
del ex
print(ex)           # name error!
```

특징3 : 딕셔너리의 key값은 중복될 수 없다.

- key값은 고유값이므로 2개 이상의 key값이 존재할 수 없다.

    - key값에 해당하는 value값을 어떤 걸 불러야할지 모르기 때문.
- key값이 중복될 경우 하나를 제외한 나머지 것들이 모두 무시된다.
    - key값에 대입한 최근의 value값을 불러온다.

Copy

```
ex = {'name': 'Groot', 'lover': 'Penguin', 'feature': 'cut
e', 'feature': 'handsome'}

print(ex)  # {'name': 'Groot', 'lover': 'Penguin', 'featur
e': 'handsome'}
```

`python에서 삼항 연산자는 어떻게 사용할 수 있나요?`

ternary operators(삼항 연산자)는 조건문을 표시하는 데 사용되는 연산자이며 [true_value] if [condition] else [false_value]의 형태로 표현된다.

Copy

```
a = 123
print("a is 123" if a==123 else "a is not 123") # a is 123

a = 456
print("a is 123" if a==123 else "a is not 123") # a is not
123
```

`*args, **kwargs의 의미와 기능은 무엇인가요?`

- args는 함수에 전달되는 argument의 수를 알 수 없거나, list나 tuple의 argument들을 함수에 전달하고자 할 때 사용한다.

파이썬에서는 어디서부터 어디까지 *args에 담아야 할지 알 수 없기 때문에, 일반 변수를 앞에 두고 그 뒤에 *args를 지정해 주어야 한다.

Copy

```
def name(*args):
    print(args)
```

```
name("샐리", "펭귄", "히스", "원딜")
# output: ('샐리', '펭귄', '히스', '원딜')
```

- *kwargs는 함수에 전달되는 keyword argument의 수를 모르거나, dictionary의 keyword argument들을 함수에 전달하고자 할 때 사용한다.
- args와 *kwargs를 함께 사용하는 경우 args를 *kwargs보다 앞에 두어야 한다.

Copy

```
def name(**kwargs):
    print(kwargs)

name(sally="샐리", penguin="펭귄", heath="히스", adc="원딜")
# output: {'sally': '샐리', 'penguin': '펭귄', 'heath': '히
스', 'adc': '원딜'}
```

일반 변수, *args, **kwargs를 모두 사용하는 경우 다음과 같은 순서를 따른다.

Copy

```
def function_name(일반변수, *args, **kwargs)
```

`len()은 어떤 기능을 하나요?`

len() 함수는 object의 길이(item의 수)를 return 한다.

argument로는 sequence(string, bytes, tuple, list, range, ...), collection(dictionary, set, frozenset, ...)을 받는다.

Copy

```
stg = "ai-tech-interview"
len(stg)  #17

ex_list = ["ai", "tech", "interview"]
len(ex_list)  # 3
```

`파이썬에서 “re” 모듈의 split(), sub(), subn() 메서드를 설명해 주시겠어요?`

파이썬에서 정규표현식을 사용하기 위해 “re” 모듈을 사용한다. 문자열 수정을 위해 Python의 “re” 모듈은 3 가지 메서드를 제공한다.

- re.split(pattern, string, maxplit=0) : pattern을 구분자로 string을 분리하여 list로 반환한다.

Copy

```
re.split('<[^<>]*>', '<html> Wow <head> header </head> <bod
y> Hey </body> </html>')
# output: ['', ' Wow ', ' header ', ' ', ' Hey ', ' ', '']
```

- re.sub(pattern, repl, string, count=0) : string에서 pattern과 일치하는 부분에 대하여 repl로 교체하여 결과 문자열을 반환한다.

Copy

```
re.sub('\d{4}', 'XXXX', '010-1234-5678')
# output: '010-XXXX-XXXX'
```

- re.subn(pattern, repl, string, count=0) : sub와 동일하나, 결과로 (결과문자열, 매칭횟수)를 튜플로 반환한다.

Copy

```
re.subn('\d{4}', 'XXXX', '010-1234-5678')
# output: ('010-XXXX-XXXX', 2)
```

`음수 인덱스의 의미와 기능은 무엇인가요?`

인덱스

- 시퀀스 객체에 [](대괄호)를 붙여 사용
- 시퀀스 객체의 인덱스는 항상 0부터 시작
- 시퀀스 객체(list, tuple, range, 문자열)에 사용가능
- 시퀀스객체[인덱스]

음수 인덱스

- 인덱스를 음수로 지정하면 뒤에서부터 요소에 접근하게 된다.
  - 1은 뒤에서 첫 번째, -5는 뒤에서 다섯 번째 요소를 뜻한다.
- 시퀀스 객체(list, tuple, range, 문자열)에 사용가능

Copy

```
example = ['Boostcamp', 'AI', 'Tech', 'penguin', 'sally',
'adc', 'heath']
print(example[5]) # adc
print(example[-2])  # adc
print(example[-4:]) # ['penguin', 'sally', 'adc', 'heath']

example = "BoostDevs"
print(example[:5])  # Boost
print(example[-4:-1]) # Dev
```

`파이썬 패키지란 무엇인가요?`

> 패키지

패키지는 특정 기능과 관련된 여러 모듈을 묶은 것으로 패키지는 모듈에 namespace를 제공한다. 패키지는 하나의 디렉토리에 놓여진 모듈들의 집합을 가리키는데, 그 디렉토리에는 일반적으로 __init__.py 라는 패키지 초기화 파일이 존재한다.

패키지는 모듈들의 컨테이너로서 패키지 안에는 또다른 서브 패키지를 포함할 수도 있다. 파일시스템으로 비유하면 패키지는 일반적으로 디렉토리에 해당하고, 모듈은 디렉토리 안의 파일에 해당한다.

`파이썬에서 파일을 어떻게 삭제하나요?`

os 모듈을 import 한 후 os.remove() 함수를 사용하여 파일을 삭제할수있다.

Copy

```
import os
os.remove("ai-tech-interview.txt")
```

`파이썬의 기본 제공 유형은 무엇인가요?`

Python의 Built-in type은 아래와 같다.

- Integer
- Floating-point
- Complex number
- String
- Boolean
- Built-in function

> 💡 빌트인built-in이란?
> 어떤 기능이나 함수 등이 내장 또는 빌트인되어 있다는 뜻은 그것이 프로그램에서 바로 사용가능하도록 준비되어 있다는 뜻이다. 자세한 내용은 basic-terminology-in-programming - shoark7 참고할 것.

`NumPy 배열은 (중첩된) 파이썬 리스트에 비해 어떤 이점이 있나요?`

numpy array는 하나의 데이터 타입만 정의가 가능하다. Python list와 달리 다이나믹 타이핑을 지원하지 않으며, C의 Array를 사용하여 배열을 생성하기 때문에 속도가 빠르다.

Python list는 데이터 주소값을 저장하고 데이터를 가져올 때는 해당 주소에 가서 데이터를 가져온다. 반면 Numpy array는 C의 배열과 유사하여 연속된 주소를 가지고 있어 데이터를 가져올 때는 순서대로 가져오면 되기 때문에 메모리를 효율적으로 사용한다.

`파이썬 배열에 값을 추가하는 방법은 무엇인가요?`

append(), extend(), insert() 함수를 사용하여 list에 value를 추가할 수 있다.

append()

list.append(x) 형태로 사용하며, $O(1)$의 시간복잡도를 갖는다. 괄호 안에 값을 입력하면 새로운 요소를 list 맨 끝에 추가한다. 요소를 추가할 때는 객체로 추가하게 되는데, 입력한 값이 리스트와 같은 반복 가능한 iterable 자료형이더라도 객체로 저장한다.

Copy

```
nums = [1, 2, 3]
nums.append(4)
print(nums) # [1, 2, 3, 4]
```

```
nums.append([5, 6])
print(nums) # [1, 2, 3, 4, [5, 6]]
```

extend()

list.extend(iterable) 형태로 사용하며, $O(N)$ 시간복잡도를 갖는다. 입력한 iterable 자료형의 항목 각각을 list의 끝에 하나씩 추가한다. iterable 자료형으로 추가되는 것이 아니라 iterable 자료형 안에 있는 항목이 하나씩 떼어져서 추가된다. append와 동일하게 요소를 list의 끝에 추가하지만 append와 다른 점은 괄호 안에는 iterable 자료형만 올 수 있다는 것이다. iterable 자료형이 아닌 경우 TypeError가 발생한다.

Copy

```
nums = [1, 2, 3]
nums.extend([4])
print(nums) # [1, 2, 3, 4]

nums.extend([5, 6])
print(nums) # [1, 2, 3, 4, 5, 6]
```

insert()

- 시간복잡도: ``

list.insert(i, x) 형태로 사용하며, $O(N)$ 시간복잡도를 갖는다. list의 원하는 위치 i 앞에 추가할 값 x를 삽입할 수 있다. i는 위치를 나타내는 인덱스를 숫자를 입력한다. 음수를 입력하면 배열의 끝을 기준으로 처리된다. 추가할 값 x는 객체로 추가되며 iterable 자료형이더라도 객체로 저장된다.

Copy

```
nums = [1, 2, 3]
nums.insert(0, 10)
print(nums) # [10, 1, 2, 3]

nums.insert(-1, 99)
print(nums) # [10, 1, 2, 99, 3]
```

```
nums.insert(len(nums), [20, 30])
print(nums) # [10, 1, 2, 99, 3, [20, 30]]
```

+ 연산자

list_1 + list_2 형태로 사용하며, $O(1)$ 시간복잡도를 갖는다.

Copy

```
nums = [1, 2, 4, 6, 1, 5]
print(nums + [10, 9, 8, 7]) # [1, 2, 4, 6, 1, 5, 10, 9, 8,
7]
```

`파이썬 배열에 값을 제거하는 방법은 무엇인가요?`

remove(), pop() 함수를 사용하여 list에 value를 삭제할 수 있다.

remove()

remove()는 지우고자 하는 인덱스가 아닌, 값을 입력하는 방식이다. 만약 지우고자 하는 값이 리스트 내에 2개 이상이 있다면 순서상 가장 앞에 있는 값을 지우게 된다. 값을 삭제할 때 삭제된 값을 반환하지 않는다. remove()는 시간복잡도 $O(N)$를 갖는다.

Copy

```
example = [1, 2, 3, 4, 5, 1]
example.remove(1)
print(example)  # [2, 3, 4, 5, 1]
```

pop()

pop()은 리스트에서 지우고자 하는 값의 인덱스를 받아서 지우는 방식이다. 값을 삭제할 때 삭제된 값을 반환한다. 인덱스를 지정하지 않으면 리스트의 마지막 요소가 삭제되며 반환된다. pop()은 시간복잡도 $O(N)$를 갖는다.

Copy

```
example = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(example.pop())  # 10
```

```
print(example)  # [1, 2, 3, 4, 5, 6, 7, 8, 9]

example = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(example.pop(3)) # 4
print(example)  # [1, 2, 3, 5, 6, 7, 8, 9, 10]
```

del

del list[i] 형태로 사용하며, 시간복잡도 $O(N)$을 갖는다. 값을 삭제할 때 삭제된 값을 반환하지 않는다.

Copy

```
example = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
del example[7]
print(example)  # [1, 2, 3, 4, 5, 6, 7, 9, 10]

example = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
del example[7:]
print(example)  # [1, 2, 3, 4, 5, 6, 7]
```

`파이썬에 OOps 개념이 있나요?`

Python은 객체 지향 프로그래밍 언어이다. Python의 주요 OOP 개념에는 Class, Object, Method, Inheritance(상속), Polymorphism(다형성), Data Abstraction(데이터 추상화), Encapsulation(캡슐화)을 포함한다.

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

dog = Dog()
dog.speak()  # Dog barks
```

`Shallow copy와 Deep copy의 차이는 무엇인가요?`

Shallow copy는 **새로운 객체(변수)를 만든 후에 원본에 접근할 수 있는 참조(reference)를 입력**한다. 이런 경우 서로 다른 변수명이지만 본질적으로 서로 같은 대상을 의미하므로 하나의 변수 역시 수정이 된다.

가변형(mutable) 자료형에 대해서 적용이 가능하다. 가변형(mutable) 자료형은 같은 주소에서 값(value)이 변경 가능하기 때문에 얕은 복사가 가능하다. 반면 불변형(immutable) 자료형은 본질적으로 변경이 불가능하므로 재배정을 통해 변수를 바꾼다. 따라서 재배정이 이루어지므로 객체가 서로 달라진다.

Copy

```
a = [1, 2, 3, 4, 5]
b = a    # shallow copy
a[1] = 10
print(a, b) # [1, 10, 3, 4, 5] [1, 10, 3, 4, 5]
```


```
a = [1, 2, 3, 4, 5]
b = a    # shallow copy
a[1] = 10
print(a, b) # [1, 10, 3, 4, 5] [1, 10, 3, 4, 5]
```


Deep copy는 **새로운 객체(변수)를 만든 뒤에 원본의 복사본을 변수에 입력**한다. 서로 값만 같을 뿐 본질적으로 서로 다르기 때문에 한 변수가 수정될 시 다른 변수가 수정되지 않는다.

Copy

```
a = [1, 2, 3, 4, 5]
b = a[:]    # deep copy
a[1] = 10
print(a, b) # [1, 10, 3, 4, 5] [1, 2, 3, 4, 5]
```

Copy

```
import copy
a = [1, 2, 3, 4, 5]
b = copy.deepcopy(a)    # deep copy
```

```
a[1] = 10
print(a, b) # [1, 10, 3, 4, 5] [1, 2, 3, 4, 5]
```

`파이썬에서 멀티스레딩은 어떻게 이뤄지나요?`

파이썬에서 멀티 쓰레드를 구현하는 방법은 threding 모듈(High level)을 사용하거나 thread 모듈(Low level)을 사용하는 방법이 있다. 현재 thread 모듈은 deprecated 되어 threading 모듈을 사용하는 것을 권장한다.

멀티스레딩을 사용하면 당연히 속도가 빨라질 것이라 생각할 수 있지만, 파이썬의 GIL(Global Interpreter Lock) 정책으로 인해, 멀티스레딩을 사용한다 하더라도, 속도는 싱글스레드와 별반 다르지 않다.

GIL은 한 번에 하나의 스레드만 실행되도록 제한한다. 따라서 CPU 중심 작업에서는 멀티프로세싱을 사용하는 것이 적합하며, I/O 작업에서는 멀티스레딩이 효과적이다.

하나의 자원에 여러 프로세스가 아무런 규칙없이 접근하면, 자원 동기화 문제가 발생할 수 있다. 이를 방지하기 위해서 자원에 lock을 두는데, Python은 모든 자원의 lock을 global 하게 관리하고 있다. 한번에 하나의 스레드만 자원에 접근할 수 있다는 것이다. 이로인해, 자원을 공유하는 여러 스레드를 동시에 실행시킨다고 해도, 결국 GIL 때문에 한번에 하나의 스레드만 실행되는 것이다.

> 💡 멀티스레딩이 유용한 경우
> GIL은 cpu 동작에 대해서만 적용된다. 쓰레드가 cpu 동작을 마치고 I/O 작업을 실행하는 동안에는 다른 쓰레드가 cpu 동작을 동시에 실행할 수 있다. 따라서 cpu 동작이 많지 않고 I/O동작이 더 많은 프로그램에서는 멀티 쓰레드만으로 성능적으로 큰 효과를 얻을 수 있다.

`파이썬에서 컴파일과 링크의 과정은 무엇인가요?`

파이썬 파일(.py)를 실행하면, 소스 코드는 바이트 코드(byte code)로 변환되며, .pyc, .pyo 파일 형식으로 저장된다. 이 때 소스 코드를 바이트 코드로 변환하는 과정을 컴파일(compilation) 단계라고 한다.

파이썬 가상머신(Python Virtual Machine)이 바이트 코드를 기계어(machine code)로 변환하여 어떤 운영체제든 실행할 수 있도록 한다. 이 때 우리의 코드와 인터프리터가 필요한 라이브러리를 연결시키는 과정이 있는데, 이를 **링크(linking) 단계**라고 한다.

참고로 dis 모듈을 사용하여 소스 코드가 어떤 바이트 코드로 변환되는지 확인할 수 있다.

Copy

```
import dis

def mult(a, b):
    return a*b

dis.dis(mult)
# output:
# 4           0 LOAD_FAST                0 (a)
#             2 LOAD_FAST                1 (b)
#             4 BINARY_MULTIPLY
#             6 RETURN_VALUE
```

`파이썬 라이브러리가 무엇인지 몇 가지 말씀해 주시겠어요?`

파이썬 라이브러리는 패키지의 모음이다.

주로 사용되는 파이썬 라이브러리로는 Numpy, Pandas, Matplotlib, Scikit-learn 등이 있다.

`split()은 언제 사용되나요?`

split()은 특정 문자를 기준으로 문자열을 분리할 때 사용한다.

Copy

```
str.split(sep=None, maxsplit=-1)
```

sep을 구분자로 사용하여 문자열에 있는 단어 list를 반환한다.

sep이 지정되면 구분자를 기준으로 문자열을 분리하고, sep이 지정되지 않았거나 None인 경우에는 whitespace를 기준으로 문자열을 분리한다.

maxsplit이 지정되면 그 수만큼의 분할이 수행되고, maxsplit이 지정되지 않았거나 -1인 경우에는 가능한 모든 분할이 수행된다.

Copy

```
a = "ai tech interview"
print(a.split())  # ['ai', 'tech', 'interview']
a = "ai                  tech          interview"
print(a.split())  # ['ai', 'tech', 'interview']
```

```
a = "ai-tech-interview"
print(a.split("-")) # ['ai', 'tech', 'interview']
a = "ai-tech-interview"
print(a.split("-", 1))  # ['ai', 'tech-interview']
```

`파이썬에서 모듈을 임포트하는 방법은 무엇인가요?`

Copy

```
import [패키지/모듈]
from [패키지] import [모듈/변수/함수/클래스]
from [모듈] import [변수/함수/클래스]
```

import키워드를 사용하여 모듈을 가져올 수 있다. 세 가지 방법으로 모듈을 가져올 수 있다.

Copy

```
import numpy         # importing using the original module n
ame
import numpy as np   # importing using an alias name
from numpy import * # imports everything present in the num
py module
```

Copy

```
from numpy import argmax as arm # 이런거도 된다
nums = [1, 2, 3, 4, 5]
print(arm(nums)) # 4
```

`파이썬에서 상속을 예시를 들어 설명해 주시겠어요?`

상속을 통해 상위 (부모) 클래스의 멤버 함수, 멤버 변수들을 모두 하위 (자식) 클래스가 가질 수 있다. 상위 클래스를 상속함으로써 코드 재사용성이 더 좋아지고, 관리가 용이해진다.

파이썬은 부모 클래스 A 를 자식 클래스 B 가 상속하는 **Single Inheritance**, 부모 클래스 A 를 자식 클래스 B 가 다시 B 를 자식 클래스 C 가 상속하는 **Multi-level Inheritance**, 부모 클래스 A 가 여러 자식 클래스에 상속되는 **Hierarchical Inheritance**, 하나의 자식 클래스가 여러 부모 클래스를 상속하는 **Multiple Inheritance** 가 있다.

`파이썬에서 클래스는 어떻게 생성되나요?`

class 키워드를 사용하여 클래스를 만들 수 있다. 이 때, 클래스명 옆 괄호에 상속받을 부모 클래스를 설정할 수도 있다.

기본적으로 __init__ 이라는 매직 메소드를 통해 멤버 변수들을 세팅할 수 있다. 자세히는 클래스가 객체로 선언될 때, 멤버 변수의 값을 초기화하는 역할을 담당한다.

클래스 내에서는 멤버 함수를 만들 수 있고, 클래스 객체에서 멤버 함수를 사용할 수 있다.

Copy

```
class MyClass():
    def __init__(self, feature):
        self.feature = feature
        ...

    def my_method(self):
        ...
```

`파이썬에서 monkey patching이란 무엇인가요?`

주로 테스트를 위해 많이 사용되는 방법으로, 어떤 클래스나 모듈의 일부 (함수나 변수 등)를 로컬에서 런타임으로만 instance 를 통해 수정하는 방법을 말한다.

예시로 heath.py 파일의 A 클래스에 a 라는 함수가 있는데, 다른 파일에서 A 를 import 하여 a 함수 대신 new_a 를 할당하여 사용하는 방법이 있다.

Copy

```
from heath import A

A.a = new_a
my_A = A() # A 클래스 객체 할당
my_A.a # new_a 가 동작
```

`파이썬은 다중 상속을 지원하나요?`

파이썬은 자바와 다르게 multiple inheritance 을 지원한다. 예시는 아래와 같다.

Copy

```
class P_A():
    ...

class P_B():
    ...

class C(P_A, P_B): #P_A 와 P_B 클래스를 동시에 상속
    ...
```

`파이썬에서 다형성이란 무엇인가요?`

다형성은 객체지향의 주요 개념으로 여러가지 형태를 가질 수 있는 능력을 말한다. 다형성은 코드의 유지보수에 도움을 준다.

파이썬은 다형성을 지원하는데, + 연산이나 len 연산에 대해 생각해볼 수 있다. 이들은 여러 타입의 변수에 대해서도 동일한 기능을 제공하는데 overriding 과 overloading 을 통해 각기 다른 타입의 변수에도 반응하도록 다형성을 주었기 때문에 가능하다.

`파이썬에서 캡슐화는 어떻게 정의하나요?`

캡슐화는 주요 변수나 함수를 외부로부터 보호하는 방법을 말한다. 캡슐화를 통해 코드의 안전성을 높일 수 있다.

파이썬에서는 클래스를 생각해볼 수 있다. 클래스의 멤버 변수나 멤버 함수에 접근하기 위해서는 클래스에 대한 객체를 만들어야 한다. 객체를 통해 멤버에 접근하기 때문에 직접 변수를 손대는 것보다 데이터를 더 안전하게 지킬 수 있다.

`파이썬에서 데이터 추상화는 어떻게 하나요?`

데이터 추상화는 객체지향의 주요 개념으로 사용자에게 데이터의 주요 정보만 제공하여 구체적인 구현은 몰라도 사용할 수 있게 만드는 방법이다.

파이썬에서는 abstract class 를 통해 데이터 추상화를 할 수 있다. abstract class 를 사용하기 위해서는 abc모듈을 import 하고 metaclass=ABCClass 와 @abstractmethod 를 사용해야 한다.

Copy

```
from abc import *
class 추상클래스명(metaclass=ABCMeta):
```

```
    @abstractmethod
       def 추상메소드(self):
           pass
```

`파이썬은 액세스 지정자를 사용하나요?`

파이썬은 다른 언어와 달리 private, protected 등의 접근 제한자를 직접 명시하지 않고 변수명을 통해 접근 제어를 한다.

접두사 _ 가 한 개 있는 경우에는 protected, 접두사 _ 가 두 개 있는 경우에는 private, 접두사 _ 가 없거나 접미사 _ 가 두 개 이상 있는 경우에는 public 이다.

`파이썬에서 빈 클래스를 생성하는 방법은 무엇인가요?`

파이썬에서 클래스 내부에 아무 내용 없이 선언만 하기 위해서는 pass 나 ... 을 사용할 수 있다. 추가적으로 empty class 를 선언한 후, 외부에서 객체를 통해 클래스의 변수나 함수를 만들 수도 있다.

Copy

```
class empty():
    ... # or pass

e = empty()
e.a = 10
print(e.a) # 10
```

`object()의 기능은 무엇인가요?`

파이썬은 모든 것이 객체이다. 따라서 기본적으로 object 클래스를 상속받고 있다. object() 함수를 사용하면 새로운 기본 object 객체를 반환받을 수 있다.

`파이썬의 map 함수란 무엇인가요?`

map 함수는 iterable 한 객체의 모든 원소에 동일한 함수를 적용하는 기능을 한다.

첫 인자로 적용할 함수를, 두번째 인자로 iterable 한 객체를 넣으면, iterable 한 map 객체 형태로 각 원소에 대해 함수가 적용된 묶음들이 담겨 나온다.

Copy

```
int_arr = list(map(int, input().split()))
```

`파이썬에서 Numpy는 list보다 좋나요?`

파이썬의 리스트는 각 원소들의 값을 직접 사용하지 않고 원소들의 주소를 참조하는 방식을 사용하기 때문에 원소들의 타입이 정해지지 않아 편리하지만 메모리를 많이 사용하고 느리다는 단점이 있다.

반면, 넘파이는 C 기반으로 구현되어 원소들의 타입을 미리 설정하여 메모리를 적게 사용하고 빠르다. 또한 행렬과 선형대수에 편리한 함수들을 제공한다는 장점도 있다.

`파이썬에서 GIL은 무엇인가요?`

멀티쓰레딩을 할 때, 공유 자원에 대해 여러 쓰레드가 동시에 접근한다면 갱신된 내용이 유실되는 등의 문제가 발생할 수 있다. 이를 막기 위해 파이썬은 GIL (Global Interpreter Lock) 을 통해 python interpreter 에 한 쓰레드만 접근하여 모든 자원을 사용할 수 있게 한다.

정확히는 멀티 쓰레드가 bytecode(=instruction) 한 라인씩을 들고 있기 때문에, 한 쓰레드의 bytecode 한 줄에 대해서만 GIL 은 허용한다.

`CPython이 Python과 다른 점은 무엇인가요?`

파이썬은 일반적으로 C 로 구현된 인터프리터 언어이다. 일반적인 C 언어와 구분하기 위해 파이썬 구현체 C 를 CPython 이라고 부른다.

CPython 은 인터프리터이면서 컴파일러로 Python 코드를 C 가 아닌 bytecode 로 컴파일해주고, 이를 interpreter(virtual machine) 가 실행하게 만든다. 이러한 CPython 의 interpreter 적인 특징이 파이썬을 차별되게 만들었다.

`파이썬에서 Decorators란 무엇인가요?`

함수를 인자로 받고 내부 함수에서 인자로 받은 함수를 사용하는 클래스나 함수가 있을 때, 인자로 사용할 함수를 간편하게 지정해주는 역할을 하는 것이 Decorator 이다.

Decorator 의 사용 문법은 인자가 될 함수 위에 **@외부함수이름** 을 붙여주면 된다.

아래 예시를 보면, Decorator 를 통해 big_number 와 big_number2 라는 서로 다른 함수를 make_time_checker 가 인자로 받아 내부 함수에서 사용하고 있다.

Copy

```
import time


def make_time_checker(func):
      def new_func(*args, **kwargs):
            start_time = time.perf_counter()
            result = func(*args, **kwargs)
            end_time = time.perf_counter()
            print('실행시간:', end_time - start_time)
            return result
      return new_func

@make_time_checker
def big_number(n):
      return n ** n ** n

@make_time_checker
def big_number2(n):
      return (n+1) ** (n+1) ** (n+1)
```

`object interning은 무엇인가요?`

파이썬에서는 모든 것이 객체이므로 변수들은 값을 바로 가지지 않고 값을 가진 주소를 참조하게 된다.

object interning 은 자주 사용될, 즉 재활용될 object 에 대해 매번 새로운 주소를 할당하는 것은 비효율적이므로, 하나의 주소에 값을 주고 그 주소를 재활용하는 작업을 말한다.

기본적으로 파이썬에서는 -5~256, [a-Az-Z0-9_] 에 대해 고정된 주소를 할당하여 interning 을 하고 있다.

`@classmethod, @staticmethod, @property란 무엇인가요?`

@classmethod

클래스 내부의 함수 중에 @classmethod 로 선언된 함수에 대해서는 클래스의 객체를 만들지 않고도 바로 접근이 가능하다. 하지만 함수의 첫 인자로 클래스를 받아서, 상속되었을 때 자식 클래스의 데이터를 따르는 특징이 있다.

@staticmethod

@staticmethod 는 @classmethod 와 마찬가지로 클래스의 객체를 만들지 않고도 바로 접근할 수 있다. 하지만 클래스를 인자로 받지 않기 때문에, 상속되었을 때에도 자식 클래스의 데이터를 따르지 않고 처음에 클래스에서 선언한 데이터대로 함수가 사용된다.

Copy

```
class Language:
    default_language = "English"

    def __init__(self):
        self.show = '나의 언어는' + self.default_language

    @classmethod
    def class_my_language(cls):
        return cls()

    @staticmethod
    def static_my_language():
        return Language()

    def print_language(self):
        print(self.show)


class KoreanLanguage(Language):
    default_language = "한국어"


>>> from language import *
>>> a = KoreanLanguage.static_my_language()
>>> b = KoreanLanguage.class_my_language()
>>> a.print_language()
나의 언어는English
>>> b.print_language()
나의 언어는한국어
```

@property

객체지향 언어에서는 캡슐화를 위해 변수를 직접 지정하지 않고 객체의 함수를 통해 지정하고 받아오는 setter, getter 함수를 사용한다. 파이썬에서는 이를 편하게 사용할 수 있도록 @property 를 제공한다.

getter 가 될 함수에 @property 를, setter 가 될 함수에 @변수.setter 를 붙이면 된다.

Copy

```
class Test:

    def __init__(self):
        self.__color = "red"

    @property
    def color(self):
        return self.__color

    @color.setter
    def color(self,clr):
        self.__color = clr

if __name__ == '__main__':

    t = Test()
    t.color = "blue"

    print(t.color)
```