Python 치트시트

1. 기본 데이터 타입 및 구조

- 숫자형: `int`, `float`, `complex` (복소수)
- 시퀀스 타입:
  - `str`: 불변(immutable) 문자열. `f"name: {name}"` (f-string), `"a" + "b"` (연결), `"a" * 3` (반복).
  - `list`: 가변(mutable) 리스트. `[1, "apple", 3.5]`
  - `tuple`: 불변(immutable) 튜플. `(1, "apple", 3.5)`
- 매핑 타입:
  - `dict`: 키-값 쌍. `{"key": "value", "name": "John"}`
- 집합 타입:
  - `set`: 중복 없는 순서 없는 컬렉션. `{1, 2, 3}`. 합집합(`|`), 교집합(`&`), 차집합(`-`).
  - `frozenset`: 불변 집합.

2. 제어 흐름

- if-elif-else: 조건문.
- for 루프:
  - `for item in iterable: ...`
  - `for i, value in enumerate(my_list): ...`
- while 루프: `while condition: ...`
- 루프 제어: `break` (종료), `continue` (건너뛰기), `else` (루프가 정상적으로 완료됐을 때 실행).
- try-except-else-finally: 예외 처리.

```
try:
    # 실행할 코드
    result = 10 / x
except ZeroDivisionError as e:
    print(f"Error: {e}")
except TypeError:
    print("Type error!")
else:
    print("No errors occurred.")
finally:
    print("This always runs.")
```

- with 문: 컨텍스트 관리자. 파일, 락 등 자원을 안전하게 사용하고 자동 해제. `with open("file.txt", "r") as f: ...`

3. 함수

- 정의: `def func_name(pos_arg, key_arg="default"): ...`
- 인수 종류:
  - 위치 인수 (Positional): 순서대로 전달.
  - 키워드 인수 (Keyword): `name=value` 형태로 전달.
  - 기본값 인수 (Default): 호출 시 생략 가능.
  - 가변 위치 인수 (`*args`): 여러 위치 인수를 튜플로 묶어 받음.
  - 가변 키워드 인수 (`**kwargs`): 여러 키워드 인수를 딕셔너리로 묶어 받음.
- 람다 함수 (Lambda): 한 줄로 된 익명 함수. `lambda args: expression`
- 타입 힌트 (Type Hints):

```
def greet(name: str) → str:
    return f"Hello, {name}"
```

- 데코레이터 (Decorators): 함수를 수정하지 않고 기능을 추가하는 함수. `@` 구문 사용.

```
def my_decorator(func):
    def wrapper(*args, **kwargs):
        print("Something is happening before the function is called.")
        result = func(*args, **kwargs)
        print("Something is happening after the function is called.")
        return result
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")
```

4. 컴프리헨션 및 제너레이터

- 리스트 컴프리헨션: `[expression for item in iterable if condition]`
- 딕셔너리 컴프리헨션: `{key_expr: val_expr for item in iterable if condition}`
- 집합 컴프리헨션: `{expression for item in iterable if condition}`
- 제너레이터 표현식: `(expression for item in iterable if condition)`
  - 메모리를 효율적으로 사용. 한 번에 하나의 항목만 생성.
- 제너레이터 함수: `yield` 키워드를 사용하여 함수를 제너레이터로 만듦.

```
def count_up_to(max):
    count = 1
    while count
```

  - 제거: `pip uninstall`
  - 목록: `pip list`
  - 요구사항 파일로 설치: `pip install -r requirements.txt`
  - 요구사항 파일 생성: `pip freeze > requirements.txt`

8. 데이브차일드의 치트 시트

시스템 관련 변수

- `argv`: 명령줄 인수
- `builtin_module_names`: 연결된 C 모듈 목록
- `byteorder`: 네이티브 바이트 순서
- `check_interval`: 신호 체크 주기
- `exec_prefix`: 루트 디렉토리
- `executable`: 실행 파일 이름
- `exitfunc`: 종료 함수 이름
- `modules`: 로드된 모듈 목록
- `path`: 모듈 검색 경로
- `platform`: 현재 플랫폼
- `stdin`, `stdout`, `stderr`: 입출력용 파일 객체
- `version_info`: 파이썬 버전 정보
- `winver`: 버전 번호

sys.argv

- `sys.argv[0]`: foo.py
- `sys.argv[1]`: bar
- `sys.argv[2]`: -c
- `sys.argv[3]`: qux
- `sys.argv[4]`: –h

sys.argv 예시: `$ python foo.py bar -c qux --h`

운영체제 관련 변수

- `altsep`: 대체 구분자
- `curdir`: 현재 디렉토리 문자열
- `defpath`: 기본 검색 경로
- `devnull`: 널 디바이스 경로
- `extsep`: 확장자 구분자
- `linesep`: 줄 구분자
- `name`: 운영체제 이름
- `pardir`: 부모 디렉토리 문자열
- `pathsep`: 경로 구분자
- `sep`: 경로 구분자

등록된 OS 이름 예: `"posix"`, `"nt"`, `"mac"`, `"os2"`, `"ce"`, `"java"`, `"riscos"`

특별한 상황에 자동 호출 메서드

- `__new__(cls)`: 인스턴스 생성 메서드, 객체가 만들어질 때 호출됨
- `__init__(self, args)`: 생성자, 객체 초기화 시 호출됨
- `__del__(self)`: 소멸자, 객체가 삭제될 때 호출됨
- `__str__(self)`: 사용자 친화적 문자열 표현 반환
- `__repr__(self)`: 개발자용 문자열 표현 반환
- `__eq__(self, other)`: 같음 비교 연산자 (==)
- `__ne__(self, other)`: 같지 않음 비교 연산자 (!=)
- `__lt__(self, other)`: 작음 비교 연산자 (<)
- `__le__(self, other)`: 작거나 같음 비교 연산자 (⩽)
- `__gt__(self, other)`: 큼 비교 연산자 (>)
- `__ge__(self, other)`: 크거나 같음 비교 연산자 (⩾)
- `__cmp__(self, other)`: 전체 비교 (파이썬 2에서 사용)
- `__hash__(self)`: 해시 값 반환 (딕셔너리 키 등에 사용)
- `__index__(self)`: 정수 인덱스 반환 (슬라이싱 등에 사용)
- `__nonzero__(self)`: 불리언 평가 (파이썬 2)
- `__getattr__(self, name)`: 존재하지 않는 속성 접근 시 호출
- `__getattribute__(self, name)`: 모든 속성 접근 시 호출
- `__setattr__(self, name, attr)`: 속성 설정 시 호출
- `__delattr__(self, name)`: 속성 삭제 시 호출
- `__call__(self, *args, **kwargs)`: 인스턴스를 함수처럼 호출할 때 동작

리스트 조작에 쓰이는 주요 메서드

- `append(item)`: 리스트 끝에 항목 추가
- `pop(position)`: 위치의 항목 제거 및 반환 (기본은 마지막 항목)
- `count(item)`: 특정 항목 개수 세기
- `remove(item)`: 특정 항목 첫 번째 등장 제거

- `extend(list)`: 리스트 뒤에 다른 리스트 내용 추가
- `reverse()`: 리스트 뒤집기
- `index(item)`: 항목의 첫 위치 반환
- `sort()`: 리스트 정렬
- `insert(position, item)`: 특정 위치에 항목 삽입

문자열 처리용 주요 메서드

- `capitalize()`: 첫 글자를 대문자로 변환
- `lstrip()`: 왼쪽 공백 제거
- `center(width)`: 문자열을 지정된 너비로 가운데 정렬
- `partition(sep)`: 구분자 기준으로 나누기
- `count(sub, start, end)`: 부분 문자열 개수 세기
- `replace(old, new)`: 문자열 치환
- `decode()`: 바이트를 문자열로 변환 (Python 2)
- `rfind(sub, start, end)`: 오른쪽에서부터 부분 문자열 찾기
- `encode()`: 문자열을 바이트로 변환
- `rindex(sub, start, end)`: 오른쪽에서부터 부분 문자열 위치 반환
- `endswith(sub)`: 특정 문자로 끝나는지 확인
- `rjust(width)`: 오른쪽 정렬
- `expandtabs()`: 탭을 공백으로 변환
- `rpartition(sep)`: 오른쪽부터 구분자 기준 나누기
- `find(sub, start, end)`: 부분 문자열 찾기
- `rsplit(sep)`: 오른쪽부터 문자열 나누기
- `index(sub, start, end)`: 부분 문자열 위치 반환
- `rstrip()`: 오른쪽 공백 제거
- `isalnum()`: 알파벳 또는 숫자인지 여부
- `split(sep)`: 구분자로 문자열 나누기
- `isalpha()`: 알파벳만 있는지 여부
- `splitlines()`: 여러 줄 문자열을 줄 단위로 분리
- `isdigit()`: 숫자인지 여부
- `startswith(sub)`: 특정 문자로 시작하는지 확인
- `islower()`: 모두 소문자인지 여부
- `strip()`: 양쪽 공백 제거
- `isspace()`: 공백 문자 여부
- `swapcase()`: 대소문자 바꾸기
- `istitle()`: 제목 문자열인지 확인
- `title()`: 각 단어 첫글자 대문자로 변환
- `isupper()`: 모두 대문자인지 여부
- `translate(table)`: 문자 매핑 변환
- `join()`: 리스트 등을 문자열로 연결
- `upper()`: 모두 대문자로 변환
- `ljust(width)`: 왼쪽 정렬
- `zfill(width)`: 숫자 문자열 앞에 0 채우기
- `lower()`: 모두 소문자로 변환

파일 객체 주요 메서드

- `close()`: 파일 닫기
- `readlines(size)`: 여러 줄 읽기
- `flush()`: 버퍼 비우기
- `seek(offset)`: 파일 위치 이동
- `fileno()`: 파일 기술자 반환
- `tell()`: 현재 파일 위치 반환
- `isatty()`: 터미널 연결 여부 확인
- `truncate(size)`: 파일 크기 조절
- `next()`: 다음 줄 읽기 (이터레이터)
- `write(string)`: 문자열 쓰기
- `read(size)`: 지정한 바이트 수 읽기
- `writelines(list)`: 여러 줄 쓰기
- `readline(size)`: 한 줄 읽기

리스트 인덱싱 및 슬라이싱 예시

예시 리스트 `a = [0,1,2,3,4,5]`

- `len(a)`: 6 (길이)
- `a[0]`: 0 (첫 요소)
- `a[5]`: 5 (마지막 요소)
- `a[-1]`: 5 (뒤에서 첫 요소)
- `a[-2]`: 4 (뒤에서 두 번째 요소)
- `a[1:]`: [1, 2, 3, 4, 5] (1번째부터 끝까지)
- `a[:5]`: [0, 1, 2, 3, 4] (처음부터 5번째 앞까지)
- `a[:-2]`: [0, 1, 2, 3] (뒤에서 두 번째 앞까지)
- `a[1:3]`: [1, 2] (1 2번째 요소)
- `a[1:-1]`: [1, 2, 3, 4] (1번째부터 뒤에서 첫 번째 앞까지)
- `a[::-1]`: [5, 4, 3, 2, 1, 0] (역순)
- `a[::-2]`: [5, 3, 1] (역순으로 2칸씩 건너뛰기)
- `b = a[:]`: 얕은 복사 (shallow copy)

날짜 및 시간 처리 메서드

- `today()`: 오늘 날짜 반환
- `fromordinal(ordinal)`: 지정된 일자(정수)로 날짜 생성
- `now(timezoneinfo)`: 현재 날짜 및 시간 반환
- `combine(date, time)`: 날짜와 시간 결합
- `utcnow()`: UTC 기준 현재 날짜 및 시간
- `strptime(date, format)`: 문자열을 날짜로 변환
- `fromtimestamp(timestamp)`: 타임스탬프로부터 날짜 생성
- `utcfromtimestamp(timestamp)`: UTC 타임스탬프로부터 날짜 생성

시간 객체 관련 메서드

- `replace()`: 시간 객체 일부 값 변경
- `utcoffset()`: UTC 오프셋 반환
- `isoformat()`: ISO 8601 형식 문자열 반환
- `dst()`: 일광 절약 시간 반환
- `__str__()`: 문자열 표현 반환
- `tzname()`: 시간대 이름 반환
- `strftime(format)`: 문자열로 시간 포맷팅

날짜 포멧팅

- `%a`: 약식 요일명 (Sun)
- `%A`: 요일명 (Sunday)
- `%b`: 약식 월 이름 (Jan)
- `%B`: 월 이름 (January)
- `%c`: 날짜와 시간
- `%d`: 일 (앞자리 0 포함) (01 31)
- `%H`: 24시간 (앞자리 0 포함) (00 23)
- `%I`: 12시간 (앞자리 0 포함) (01 12)
- `%j`: 연중 일자 (001 366)
- `%m`: 월 (01 12)
- `%M`: 분 (00 59)
- `%p`: 오전/오후 (AM/PM)
- `%S`: 초 (00 61)
- `%U`: 주 번호(00 53, 일요일 시작)
- `%w`: 요일(0=일요일, 6=토요일)
- `%W`: 주 번호(00 53, 월요일 시작)
- `%x`: 날짜
- `%X`: 시간
- `%y`: 연도 (세기 제외, 00 99)
- `%Y`: 연도 (예: 2008)
- `%Z`: 시간대 (GMT 등)
- `%%`: 리터럴%문자