dk techin
a kakao company

Python 문법

dktechin
a kakao company

파이썬에 대하여

- 파이썬의 장점

1. 입문자에게 많이 추천되는 언어로 배우기 쉬운 문법.
2. 가독성이 높으며 생산성이 높음.
3. 생태계가 잘 구축되어있어 다양한 분야에서 활용되며 폭넓게 사용.
   (웹, 스크립트, 데이터분석 등)

- 파이썬의 단점

1. 인터프리터 언어로 실행 될 때마다 번역이 이루어져 실행속도가 느림.
2. 파이썬은 동적 타입 언어로 변수의 타입을 정의 할 수 없어 프로그램의 규모가 커질수록 버그 및 에러를 잡아내기 힘듬.

목차

dktechin
a kakao company

1. 언어의 이해

1. 언어의 이해

세부 목차

컴퓨터 언어	• 자연어 • 기계어 • 어셈블리어 • 컴파일 언어 • 인터프리터 언어
언어의 타입	• 강한 언어 • 약한 언어 • 고수준 언어 • 저수준 언어

1. 언어의 이해

1. 컴퓨터 언어

자연어

- 인간이 사용하는 언어.
- 서로 의사소통을 하기 위한 모든 언어 체계.
- 매우 다양하고 유연한 구조.

자연어 예시

➢ 디스크가 장애 인가요?
➢ 아니면 컨트롤러가 장애인가요?

기계어

- 컴퓨터가 직접 이해하고 실행할 수 있는 유일한 언어.
- 컴퓨터가 직접 이해하고 실행할 수 있는 유일한 언어.
- 모든 프로그래밍 언어는 최종적으로 기계어로 변환되어 해석.
- 0과 1로 이루어져 있는 단순한 구조. (이진수)

기계어 예시

➢ 10110000 00000001 00000010

어셈블리어

- 모든 프로그래밍 언어는 최종적으로 기계어로 변환되어 해석.
- 기계어를 사람이 이해 및 작성이 가능하도록 기계어와 1:1 대응시킨 언어
- 어셈블리어는 컴퓨터의 하드웨어를 직접 제어하기 위해 사용되는 언어
- CPU 아키텍처마다 지원하는 고유한 어셈블리어 코드가 다름

어셈블리어 예시

➢ MOV R1, 3 // 3을 R1에 저장 .
➢ MOV R2, 4 // 3을 R1에 저장.
➢ ADD R1, R2 // R1과 R2 를 더한 후 R1에 저장.

1. 언어의 이해

1-1. 컴퓨터 언어

■ 컴파일러

- 사람이 작성한 코드를 컴퓨터가 이해 가능한 기계어로 번역하는 과정을 거쳐서 실행되는 프로그래밍 언어.
- 생성한 전체 프로그램 코드를 스캔하여 한꺼번에 번역하는 과정을 거쳐 실행. 해당 과정을 통해 실행속도가 빠름.
- 대표적인 컴파일러로 C, C++, C# JAVA 속함.

■ 해석 순서

➢ C, C++해석 순서
- 소스코드 -> 컴파일러 -> 기계어 ->실행

➢ C#, Java 해석 순서
- 소스코드 -> 컴파일러 -> 중간언어코드 -> JIT컴파일러 >기계어 -> 실행 (C#, Java 의 경우 JIT컴파일러 가상VM 이라는 것이 존재) (컴파일러와 인터프리터의 하이브리드)

■ 인터프리터

- 인터프리터는 코드를 한줄 씩 읽어가며 실행하는 프로그래밍 언어.
- 통역사가 말을 즉시 번역 해주듯 인터프리터는 소스코드를 기계어로 변환 과정 없이 바로 실행. 해당과정을 통해 실행속도가 느림
- 실행 전에 코드의 오류나 에러가 있을 시 잡아 낼 수 없는 것이 단점.
- 대표적인 인터프리터로 Python, JavaScript 가 속함

■ 해석 순서

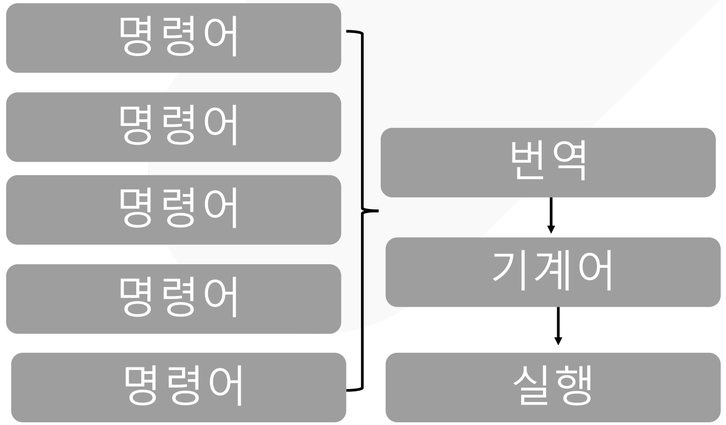
➢ 인터프리터 해석 순서
- 소스코드 -> 인터프리터 ->기계어 ->실행

■ 예제

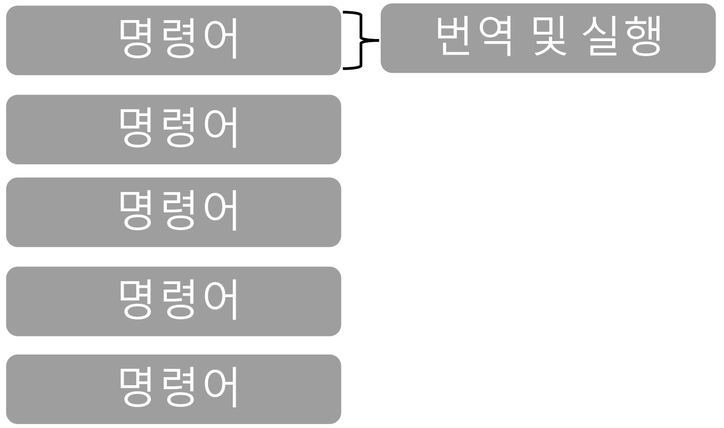
➢ company = “techin”
print(abs(campany)) // 실행 후 에러 발생

- 어떤 숫자를 대입 시 abs함수는 절대값을 반환 하는데 techin이라는 문자열에도 실행 전까지 에러를 유무 확인 불가.

■ 컴파일러



■ 인터프리터



1. 언어의 이해

2. 언어의 타입

구분	저수준 언어	고수준 언어
접근성	컴퓨터가 이해 하기 쉬운 언어 (기계어)	인간이 이해하기 쉬움 (Java, Python)
학습 난이도	배우기 어려움	배우기 쉬움
코드 가독성	매우 낮음	높음
이식성	특정 플랫폼어 종속적	여러 플랫폼에서 사용
성능	매우 빠르고 효율적	속도가 다소 느림
사용 사례	시스템 프로그래밍, 드라이버 개발 등	응용프로그램, 웹개발 등

구분	강한 타입 언어	약한 타입 언어
변수	변수 타입 선언을 엄격하게 구분	변수 타입 선언을 관대하게 구분
선언	변수 형을 선언 후 변경 불가	변수 형을 선언 후 변경가능
타입	정적 타입 컴파일 시 변수의 타입 결정	동적 타입 실행 시 변수의 타입 결정
장점	엄격한 규칙으로 결과의 일관성 보장 및 에러 방지.	코드 작성이 간편하고 빠름 매우 유연함
단점	모든 변수 타입을 명시해야 되므로 코드가 복잡하고, 유연성 감소	실행 중 타입 오류의 위험성 증가, 코드 동작 예측이 어려워 디버깅이 어려움
대표적 사례	C, C++, C#, Java 등	Python, Javascript 등

2. 데이터 타입

2. 데이터 타입

세부 목차

2. 데이터 타입

dk techin
a kakao company

1. 데이터 타입의 종류

종류	타입	특징
숫자형	정수	정수를 표현하는 데이터 타입 , 양의 정수(1), 0, 음의정수(- 1)
	실수	소수점이 포함된 숫자를 표현하는 타입 (0.1)
	복소수	실수 부분과 허수 부분으로 이루어진 숫자를 표현하는 타입 (1 +3i)
시퀀스 자료형	리스트	여러개의 데이터를 순서대로 저장하는 데이터의 집합, 대괄호로 표현 ([])
	튜플	리스트와 유사하지만, 한 번 생성되면 값을 변경할 수 없는 불변의 데이터 구조, 순서가 있고 중복된 값 허용, 소괄호로 표현 (())
	딕셔너리	키(key)와 값(Value)의 쌍으로 이루어진 데이터를 저장하는 타입, 중괄호로 표현 ({})
	집합(Set)	중복을 허용하지 않고 순서가 없는 데이터 타입, 중괄호로 표현 ({})
이 외	문자열	텍스트로 이루어진 데이터 타입, 따옴표나 큰따옴표로 표현 (‘ ‘) (“ “)
	불리언	참(True) 또는 거짓(False)를 나타내는 데이터 타입
	NoneType	값이 없음을 나타내는 특별한 데이터 타입

2. 데이터 타입

1-2. 시퀀스자료형 – 리스트 사용법

함수	설명
append(a)	리스트에 a값을 추가
insert(index, a)	리스트 index위치에 a값을 추가
extend(a)	리스트에 a값을 연결 하여 추가
remove(a)	리스트의 특정 값 제거
del(index)	리스트의 index위치의 값을 제거
pop(a)	리스트의 마지막 값을 반환 후 삭제

함수 예시

- **append(a)**

```
a = [1, 2, 3]
a.append(4)     ->#print 결과
print(a)            [1, 2, 3, 4]
```

- **insert(index, a)**

```
a = [1, 2, 3]
a.insert(1, 4)    ->#print 결과
print(a)              [1, 4, 2, 3]
```

- **extend(a)**

```
a = [1, 2, 3]
a.extend([4, 5])    ->#print 결과
print(a)                [1, 2, 3, 4, 5]
```

- **remove(a)**

```
a = [1, 2, 3, 3, 4]
a.remove(3)     ->#print 결과
print(a)            [1, 2, 3, 4]
                    (맨 처음 3 제거)
```

- **del(a)**

```
a = [‘a’ ‘b’, ‘c’, ‘d’, ‘e’]
a.del(3)       ->#print 결과
print(a)           [a, b, c, e]
                   (3번 자리 값 제거)
```

- **pop(a)**

```
a = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
b=a.pop()      ->#print 결과
print(a)           [a, b, c, d]
print(b)           [e]
```

2. 데이터 타입

1-2. 시퀀스자료형 – 리스트 사용법

함수	설명
count(a)	리스트 안에 있는 특정 값의 개수를 반환
index(a)	특정 값의 index 위치의 값을 반환.
solt()	리스트 안에 내부 값을 오름차순으로 정리 (내부값은 int, str타입 통일)
reserve()	리스트 안의 내부 값들의 순서 뒤집기
clear()	리스트 내부의 모든 값을 삭제
copy()	리스트의 내부의 값을 복사

함수 예시

- **count(a)**

```
a = [‘a’, ‘b’, ‘b’, ‘b’, ‘c’, ‘c’, ‘d’]
print(a.count(‘b’))    ->#print 결과
                       [3]
```

- **index(a)**

```
a = [‘a’, ‘b’, ‘c’]
print(a.index(‘c’))    ->#print 결과
                       [2]
```

- **solt()**

```
a = [1, 3, 4, 2]
b = [b, 3, 1, 2]       ->#print 결과
a.solt()               a = [1, 2, 3, 4]
b.solt()               b = error
```

- **reserve()**

```
a = [1, 2, 3, 4]
a.reserve()    ->#print 결과
print(a)       [4, 3, 2, 1]
```

- **clear()**

```
a = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
a.clear()      ->#print 결과
print(a)       []
```

- **copy()**

```
a = [‘a’, ‘b’, ‘c’]
b=a.copy()     ->#print 결과
print(a)       [a, b, c]
print(b)       [a, b, c]
```

2. 데이터 타입

dktechin
a kakao company

1-2. 시퀀스자료형 – 튜플 사용법

튜플의 사용

1. 튜플은 리스트와 유사하게 여러 값을 저장하는 자료형 타입.
2. 상수 값 저장 : 변경되지 않아야 하는 값들을 저장할 때 사용.
3. 튜플은 리스트와 다르게 수정이 불가능하여 append, del과 같은 함수 사용하여 요소 변경 불가
4. 함수의 여러 반환 값 : 함수에서 여러 값을 반환할 때 튜플을 사용.

구분	튜플	리스트
변경 가능성	변경 불가능	변경 가능
선언 방법	tuple = (a, b, c)	list = (a, b, c)
사용 목적	변경되지 않는 데이터 저장	자주 변경되는 데이터 저장
메모리 사용	상대적 적음	상대적 많음

사용 예시

- **변하면 안되는 값 저장 시 예시**

```
red = ( 255, 0, 0 )
green = ( 0, 255, 0 )
blue = (0, 0, 255 )
```

- **더하기 곱하기 (빼기 지원X)**

```
tu1 = ( 2, a )
tu2 = ( 2, 3 )          ->#print 결과
print(tu1 + tu2)          (2, a, 2, 3)
print(tu1 * 2)            (2, a, 2, a)
```

- **인덱싱**

```
tu1 = (1, a, 2, b)
print(tu1[1])          ->#print 결과
                         (a)
```

2. 데이터 타입

dktechin
a kakao company

1-2. 시퀀스자료형 – 딕셔너리 사용법

사용법	설명
keys()	딕셔너리의 모든 키를 한번에 확인.
values()	딕셔너리의 모든 값을 한번에 확인.
items()	딕셔너리 키와 값을 튜플로 구성된 변수 객체로 반환.
update()	딕셔너리의 키와 값을 업데이트 진행 기존 키 덮어쓰고 없는 키는 추가.
get(key, default)	키에 해당하는 값을 반환, 해당 키가 없을 시 default 값 반환.
pop(key)	키에 해당하는 값을 삭제하고 반환.
clear()	딕셔너리 안의 모든 키와 값 삭제.
del	딕셔너리 안의 키를 삭제

함수 예시

- **key()**

```
student = {'name' : 'Alice', 'age' : 20}
print(student.keys())
                    ->#print 결과
                      name
                      age
```

- **value()**

```
student = {'name' : 'Alice', 'age' : 20}
print(student.value())
                    ->#print 결과
                      Alice
                      20
```

- **items()**

```
student = {'name' : 'Alice', 'age' : 20}
print(student.items())
                    ->#print 결과
student([('name', 'Alice'), ('age', '20')
```

- **update()**

```
student1 = {'name' : 'Alice', 'age' : 20}
student2 = { 'name' : Alice', 'score' : 80'}
student1.update(student2)
print(student1)
                    ->#print 결과
{name' : 'Alice', 'age' : 20, 'score' : 80}
```

- **get()**

```
student = {'name' : 'Alice', 'age' : 20}
                                ->#print 결과
print(student.get('name'))          Alice
print(student.get('score'))         None
print(student.get('score', 80))     80
```

- **pop()**

```
student1 = {'name' : 'Alice', 'age' : 20, 'score' : 80}
student2= student.pop('age')  ->#print 결과
print(student1)          {'name' : 'Alice', 'score' : 80}
print(student2)          20
```

- **clear()**

```
student = {'name' : 'Alice', 'age' : 20}
print(student.clear())    ->#print 결과
                            {}
```

- **del**

```
student = {'name' : 'Alice', 'age' : 20}
del student['age']
print(student)            ->#print 결과
                            {'name' : 'Alice'}
```

2. 데이터 타입

dktechin
a kakao company

1-2. 시퀀스자료형 – set 사용법

사용법	설명
add()	set 자료형에 값을 추가.
update()	set 자료형에 여러 데이터를 추가.
remove()	set 에 해당하는 데이터를 제거 없으면 Error 발생.
copy()	set 자료형에 여러 데이터를 복사
합집합	| 합집합 연산자.
교집합	& 교집합 연산자.
차집합	- 차집합 연산자.

함수 예시

- **add()**

```
set = {a, b, c}
set.add(e)
print(set)          ->#print 결과
                      {a, b, c, d}
```

- **update()**

```
set = {a, b, c}
set.update( [b, c, d, e] )
print(set)          ->#print 결과
                      {a, b, c, d, e}
```

- **remove()**

```
set = {a, b, c}
set.remove(b)
print(set)          ->#print 결과
                      {a, c}
```

- **update()**

```
set = {a, b, c}
set2 = set.copy()
print(set)
print(set2)         ->#print 결과
                      set = {a, b, c}
                      set2 = {a, b, c}
```

- **합집합, 교집합, 차집합**

```
set = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}
```

합집합

```
set3 = set | set2
print(set3)
->#print 결과 : {1, 2, 3, 4, 5, 6, 7}
```

교집합

```
set3 = set & set2
print(set3)
->#print 결과 : {3, 4, 5}
```

차집합

```
set3 = set - set2
print(set3)
->#print 결과 : {1, 2}
```

2. 데이터 타입

dktechin
a kakao company

1-2. 시퀀스자료형 – 공통기능

사용법	설명
값 in 시퀀스 객체	특정 값이 있는지 확인 하는 함수. 있으면 True 없으면 False를 반환.
더하기	시퀀스 객체를 연결하는 방법.
곱하기	시퀀스 객체 * 숫자 숫자만큼 반복해서 시퀀스 객체를 생성
슬라이싱	[시작 인덱스:끝 인덱스 : 간격] [:3] -> 처음부터 3번째 데이터 추출 [7:] ->7번째부터 끝까지 데이터 추출 [2:6] ->2번째 데이터 ~ 5번째 데이터 추출 [::2] ->처음부터 끝까지 2칸씩 건너뛰며 추출 [-1] -> 뒤에서 첫번째데이터 추출 [::-1] ->역순으로 추출

함수 예시

- **값 in 시퀀스 객체**

```
company = [‘dktechin’, ‘kakao’]

print(‘dktechin’ in company)
print(‘kakaopay’ in company)

->#print 결과
  True
  False
```

- **더하기**

```
company1 = [‘dktechin’, ‘kakao’]
company2 = [‘kakaopay’]
print(company1 + company2)

->#print 결과
  [‘dktechin’, ‘kakao’, ‘kakaopay’]
```

- **곱하기**

```
num1 = [1, 4]
num2 = num1* 3
print(num2)     ->#print 결과
                  [1, 4, 1, 4, 1, 4]
```

- **슬라이싱**

```
slice = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
                  #print 결과
print(slice[:3])    [0, 1, 2]
print(slice[7:])    [7, 8, 9]
print(slice[2:6])   [2, 3, 4, 5]
print(slice[::2])   [0, 2, 4, 6, 8]
print(slice[-1])    [9]
print(slice[::-1])  [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

```
company = ’dktechin’
                  #print 결과
print(company[-1])  n
print(company[2:])  techin
```

2. 데이터 타입

dktechin
a kakao company

2. 타입의 변환

형변환	1. 형변환은 변수에 저장된 데이터 타입을 다른 데이터타입으로 변경 하는 것. 2. 2가지로 나뉘며 암시적변환(묵시적변환), 명시적변환으로 분류

구분	암시적 변환	명시적 변환
수행 주체	파이썬 인터프리터	프로그래머
자동/수동	자동	수동
목적	데이터 손실 없이 연산	원하는 타입으로 강제 변환
주의 사항	예상치 못한 결과 발생 가능	데이터 손실 발생 가능
정의	데이터 타입이 다른 값들을 연산하거나 비교 할 때 자동으로 변환하는 것	프로그래머가 직접 데이터 타입을 변환하는 것
사용 이유	프로그래머가 매번 데이터 형을 변환하는 작업이 따로 필요 없어 편의성이 좋음. 서로 다른 데이터 형 간의 연산을 자연스럽게 수행.	암시적 변환에서는 데이터 손실의 가능성이 있어 이러한 손실을 방지 하거나 제어 하기 위해 사용.

2. 데이터 타입

dktechin
a kakao company

2. 암시적 변환

■ **암시적 변환 예시**

1. 산술 연산 – 연산시 다른 숫자형 간의 연산에서도 발생.
(가장 큰 범위 숫자형의 결과로 따라감)
ex) 정수 + 실수 = 실수

예제)

```
x = 10
y = 3.14
z = x + y    # z는 자동으로 실수 타입으로 변경 13.14
```

2. 비교 연산 – 문자열과 숫자를 비교시 문자열이 숫자형,
숫자형이 문자열로 변환되어 비교 진행

예제)

```
“3” > 2        # True ( 문자열 ”3”이 숫자 3 으로 변환되어 비교)
```

3. 논리 연산 – 숫자를 불리언 값으로 변환
0은 False / 0 제외한 값은 True로 변환

예제)

```
if 100:                    # 100은 0이 아니므로 True
print(“100은 True”)
```

4. 시퀀스 연산 – 시퀀스 자료형의 경우 문자열과 리스트가 연결됨

예제1)

```
[1, 2] + “3”        #[1, 2, ”3”]
                    문자열을 리스트에 연결 시 각 문자가 리스트 데이터로 추가
```

2. 데이터 타입

2. 명시적 변환

함수 이름	설명
int()	다른 자료형을 정수형으로 변환
float()	다른 자료형을 실수형으로 변환
str()	다른 자료형을 문자열형으로 변환
list()	다른 자료형을 리스트형으로 변환
tuple()	다른 자료형을 튜플형으로 변환
set()	다른 자료형을 집합형으로 변환
dict()	다른 자료형을 딕셔너리형으로 변환

함수 예시

- **실수 -> 정수**

```
a = 3.14
b = int(a)
print(b)        ->#print 결과
                  3
```

- **문자열 -> 실수**

```
a = ”3.14”
b = float(a)
print(b)        ->#print 결과
                  3.14
```

- **정수 -> 문자열**

```
a = 100
b = str(a)
print(b)        ->#print 결과
                  “100”
```

- **문자열 -> 리스트**

```
a = ”list”
b = list(a)
print(b)          -> #print 결과
                     [’l’, ‘l’, ’s’, ’t’]
```

- **리스트 -> 튜플**

```
a = [1, 2, 3]
b = tuple(a)
print(b)          -> #print 결과
                     (1, 2, 3)
```

- **리스트 -> 셋**

```
a = [1, 2, 3]
b = set(a)
print(b)          -> #print 결과
                     {1, 2, 3}
```

- **리스트 -> 딕셔너리**

```
a = [(”a”,1), (”b”, 2), (“c”,3)]
b = set(a)
print(b)        -> #print 결과
                   {”a” : 1, “b” : 2, “c” : 3}
```

3. 출력문

3. 출력문

세부 목차

print()	• print() 함수 • escape • separator 옵션 • end 옵션
문자열 포맷팅	• %operator • str.string • f-string

3. 출력문

dktechin
a kakao company

3-1. print

print() 함수

1. python에서 print()함수는 콘솔이나 터미널에 출력을 표시하는데 사용되는 필수적인 함수.
2. python 사용자가 코드의 결과를 확인하고 디버깅하는데 필수적인 도구.
3. print()

사용법	설명
\n	줄 바꿈
\t	문장 내에 탭 표시
\\	문장 내에 역슬래시(\)를 표시 역슬래시(\)가 하나 일때 Error 발생
\', \"	문장 내에서 따옴표, 큰따옴표 표시
\r	커서를 맨 앞으로 이동(캐리지 리턴)
\b	백스페이스(한 글자 삭제)

sep	여러 데이터 사이에 구분자로 분리해서 출력하는 옵션
end	다음 출력값을 이어서 출력

출력 예시

- **기본 출력**

```
print("Hello Dktechin!")
->#print 결과 : Hello Dktechin!
```

- **변수 출력**

```
str1 = "Hello"
str2 = "Dktechin"
print(str1 + str2)
->#print 결과 : Hello Dktechin!
```

- **end**

```
print("디케이", end="/")
print("테크인")
-> #print 결과    디케이/테크인
```

- **escape**

```
print(" 엔 \n 터 ")
-> #print 결과
엔
터

print(" 엔 \t 터 ")                    엔    터
print(" 디케이 \\ 테크인 ")             디케이\테크인
print(" |'디케이|' |"테크인 |" ")      '디케이' "테크인"
print(" 디케이 |r 테크인 ")            테크인 디케이
print(" 디케이 테크인 |b" )            디케이 테크
```

- **sep**

```
print("dk", "tech", "in")
-> #print 결과
dktechin
print("dk", "tech", "in", sep=/)
dk/tech/in
```

3. 출력문

3-2. 문자열 포맷팅

%operator

- Python 초기버전에서 사용되던 전통적인 방식
- 자료형 별로 특정 문자를 사용해 지정.
- Python 아무버전이나 사용가능

포멧 스프링	설명
%s	문자열
%d	정수
%f	실수

출력 예시

- **%operator 예제1**

```
name = ’dktechin’
age = 9
print(”Hello! my name is %s and I am %d years old” %(name, age))
```

->#print 결과 : Hello my name is dktechin and I am 9 years old

- **%operator 예제2**

```
center_name = “AY”
company = “KAKAO”
age = 71
print(”I work in the %s data center department at %s. I am %d years old.” %(center_name, company, age))
```

->#print 결과 : I work in the AY data center department at KAKAO. I am 71 years old.
위와 같이 변수가 많으면 많을수록 가독성이 떨어짐.

3. 출력문

dktechin
a kakao company

3-2. 문자열 포맷팅

str.format	• 이 방식은 중괄호{} 를 사용하여 값을 넣을 자리를 표시. • format() 메서드를 통해 실제 값을 전달 받는 방식으로 동작. • Python 2.6 이상 버전부터 사용가능

출력 예시

- **str.format 예제2**

```
name = ’monkey’
age = 10
print(”My name is {} and I am {} years old.”.format(name, age))
```

->#print 결과 : My name is monkey and I am 10 years old

- **str.format 예제2**

```
center_name = “AY”
company = “KAKAO”
name = “monkey
age = 71
print(”I work in the {} data center department at {}. My name is {} and I am {} years old.” %(center_name, company, name, age))
```

->#print 결과 : I work in the AY data center department at KAKAO. I am 71 years old.
str.format 메서드도 위와 같이 변수가 많아지면 많아질수록 가독성이 떨어짐.

3. 출력문

3-2. 문자열 포맷팅

f-string

- 이 방식은 문자열 앞에 'f' 또는 'F' 를 붙여서 사용.
- 중괄호{} 안에 변수를 직접 삽입 가능.
- 기존의 문자열 포맷팅 방식보다 간단하며 가독성이 뛰어남.
- Python 3.6 이상 버전부터 사용가능

포멧 스프링	설명
%s	문자열
%d	정수
%f	실수

출력 예시

- **%operator 예제1**

```
name = "monkey”
age = 10
print(f”Hello! my name is {name} and I am {age} years old”)
```

->#print 결과 : Hello my name is monkey and I am 10 years old

- **%operator 예제2**

```
name = “Alice”
company = “KAKAO”
age = 30
hobby = ”game”
print(f”이름 : {name}, 회사 : {company}, 나이 : {age}, 취미 : {game}”)
```

->#print 결과 : 이름 : Alice, 회사 : KAKAO, 나이 : 30, 취미 : game.

4. 입력문

4. 입력문

세부 목차

input()

- input()
- split()
- map()

4. 입력문

dktechin
a kakao company

4-1. input

input()	• 사용자로부터 키보드로 값을 입력 받는 함수 • 어떠한 변수를 사용자에게 직접 입력해서 활용될 때 사용

함수	설명
input()	입력 데이터는 항상 문자열.
split()	입력 받은 값을 구분자 기준으로 분리할 때 사용 구분자의 기준은 공백,탭, 줄 바꿈.
map()	map(함수, 객체) 여러 개의 데이터를 입력 받아 각 함수에 적용하여 결과값을 반환

예제

- **기본 출력**

```
num1 = input(“ 숫자 입력 : ”)
                                숫자 입력 : 5
print(num1)                     #print 결과 : 5
```

- **문자열로 입력**

```
num1 = input(“ 숫자 입력 : ”)    숫자 입력 : 5
num2 = input(“ 숫자 입력 : ”)    숫자 입력 : 3

print(num1 + num2)              #print 결과 : 53
```

- **문자열로 입력**

```
num1 = int(input(“ 숫자 입력 : ”))  숫자 입력 : 5
num2 = int(input(“ 숫자 입력 : ”))  숫자 입력 : 3

print(num1 + num2)              #print 결과 : 8
```

- **split()**

```
str = input(“어떤 음식이 맛있나요? : ”).split()
                        문자 입력 : 바나나가 맛있어요.
print(str)              #print 결과 : [‘바나나가’, ’맛있어요’]
```

- **map()**

```
num, num2= input(“숫자 2개 입력 : ”).split()
                        문자 입력 : 3  5
print(num + num2)       #print 결과 : 35
                        위와 같이 문자열로 저장

num, num2= map(int(input(“숫자 2개 입력 : ”).split())
                        문자 입력 : 3  5
print(num + num2)       #print 결과 : 8
                        int형태로 입력값  저장
```

5. 조건문

5. 조건문

세부 목차

if

- True, False
- if
- if ~else
- if ~elif

5. 조건문

5-1. True, False

조건문

- 파이썬에서 주어진 조건이 참인지 거짓 인지에 따라 코드의 실행을 제어
- 어떤 조건이 참(True)일시 특정 코드를 실행.
- 참이 아니고 거짓(False)일시 다른 코드 실행.

True, False, None이 True, False가 되는경우

True, False, None	설명
True	True로 평가
False	False로 평가
None	False로 평가

숫자가 True, False가 되는경우

숫자	설명
0이 아닌 숫자	True로 평가
0	False로 평가

문자열이 True, False가 되는경우

문자열	설명
모든 문자열, 공백	True로 평가
빈문자열	False로 평가

시퀀스 자료형이 True, False가 되는경우

리스트, 튜플, 딕셔너리, 집합	설명
요소가 하나이상 존재	True로 평가
빈 자료형	False로 평가

True, False 예시

```
name = “Alice”
hobby =[]

if name :       -> #True 문자열
   …
if hobby :      -> #False 빈 배열
   …
if 30 :         -> #True 모든 숫자
   …
if 0 :          -> #False 0
   …
if “” :         -> #False 공백
   …
```

5. 조건문

dktechin
a kakao company

5-2. if

조건문(if문)

- 참인 경우에만 실행되도록 하는 가장 기본적인 문법.
- if 조건식 : 로 조건식을 명시.
- 다음 줄에서 들여쓰기를 통해 코드 작성.

if 문

if 조건 표현 :
  True 일시 수행할 표현

예제

- **if 예제1**

```
age = 30
if age == 30 :
   print(“I am 30 years old”)
```

->#print 결과 : I am 30 years old

- **if 예제2**

```
age = 20
if age == 30 :
   print(“I am 30 years old”)
```

->#print 결과 : 없음.

- **if 예제3**

```
a = 10
b = 30
if a<b :
print( “10은 30 보다 작다.”)
```

->#print 결과 : I am 30 years old

- **if 예제4**

```
a = 10
b = 30
if a>b :
print( “10은 30 보다 크다.”)
```

->#print 결과 : 없음.

5. 조건문

dktechin
a kakao company

5-2. if ~else

양자 택일 조건문(if ~else문)

- 조건식이 참이 되는 이외의 모든 경우를 else문으로 실행.
- if문의 조건식이 False인 경우 else문에 담아둔 코드를 실행.

if else 문

```
if 조건 표현 :
    True 일시 수행할 표현
else :
    False 일시 수행할 표현
```

예제

- **if 예제1**

```
age = 30
if age == 30 :
    print("I am 30 years old")
else :
    print("I am not 30 years old")
```

->#print 결과 : I am 30 years old

- **if 예제2**

```
age = 20
if age == 30 :
    print("I am 30 years old")
else :
    print("I am not 30 years old")
```

->#print 결과 : I am not 30 years old .

- **if 예제3**

```
score = 70
if score >=  70 :
    print("합격입니다")
else :
    print("불합격입니다,")
```

->#print 결과 : 합격입니다.

- **if 예제4**

```
score = 60
if score >  70 :
    print("합격입니다")
else :
    print("불합격입니다,")
```

->#print 결과 : 불합격 입니다

5. 조건문

dktechin
a kakao company

5-2. if ~elif

다자 택일 조건문(if ~elif문)

- 조건식이 False인 경우 else문으로 실행되는데 이때 추가적인 조건을 걸어줘야 하는 경우 사용.
- else문에 추가로 if문을 중첩하여도 되지만 elif문으로 중첩하지 않고 깔끔하게 코드 작성 가능.

if elif 문

```
if 조건 표현 :
   A조건이 True 일시 수행할 표현
elif 조건 표현 :
   False 일시 수행할 표현
   (A조건은 거짓)
else :
   모든 조건이False 일시 수행할 표현
```

예제

- **if 예제1**

```
work = "당직 근무"

if work == "주간 근무" :
   print("AM 10시 출근!")
elif day == "오후 근무" :
   print("PM 1시 출근!")
elif day == "당직 근무" :
   print("PM 10시 출근!")
else :
   print("휴무!!")

->#print 결과 : PM 10시 출근!
```

- **if 예제2**

```
A = 80
B = 70

if A==B :
   print("A는 B와 몸무게가 같습니다.")
elif A>B :
   print("A가 B보다 몸무게가 많이 나갑니다.")
elif A!=B
   print("A는 B와 몸무게가 같지 않습니다.")
else
   print("A가 B보다 몸무게가 많이 나갑니다.")

->#print 결과 : A가 B보다 몸무게가 많이 나갑니다.
(A>B와 A!=B 모두 참이지만 if문은 True일시 문장을 실행 후 종료.)
```

6. 반복문

6. 반복문

세부 목차

while, for

- while
- for
- for in range
- for in enumerate
- 리스트 컴프리헨션

6. 반복문

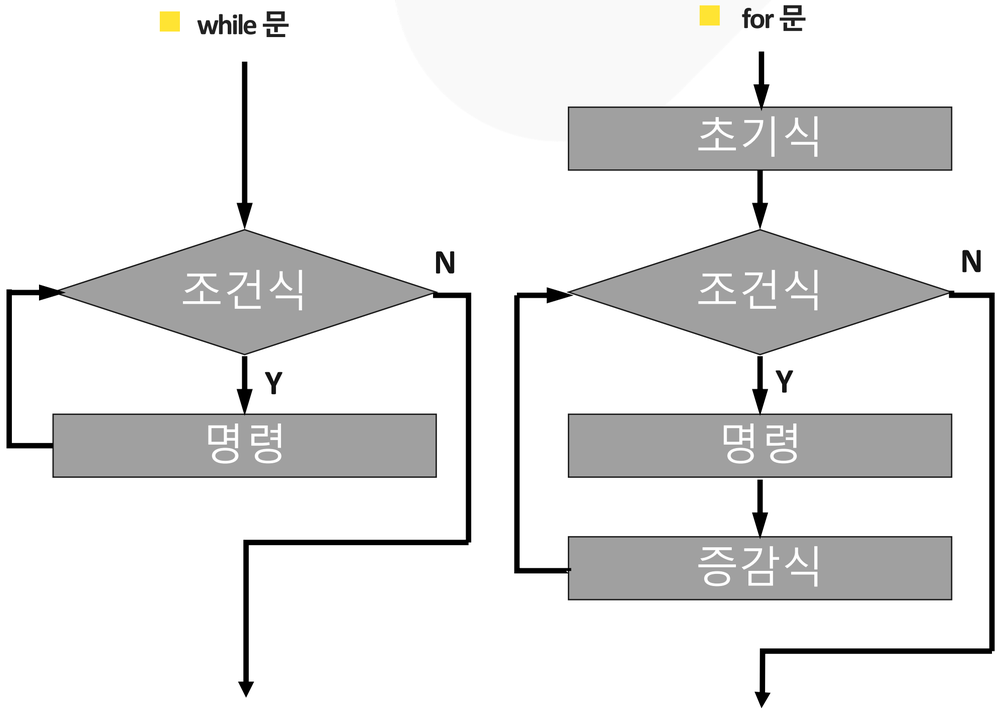
dktechin
a kakao company

6-1. 반복문

반복문

- 특정한 코드를 반복적으로 실행하고자 할 때 사용되는 문법.
- 컴퓨터에게 특정 작업을 반복 지시.
- 대표적인 반복문으로 while문, for문 두가지가 있음.

특징	while	for
반복 조건	특정 조건이 참일 경우 계속	모든 내부 요소를 순회
반복 횟수	명시된 조건이 False가 되면 종료	반복 가능한 객체나 range() 같은 내부 요소에 따라 종료
사용 목적	특정 조건이 만족할 때까지 반복	반복 횟수를 정확히 알 때 사용
반복 구조	유연한 반복 구조	구조화된 반복에 적합
모드 간결함	상대적으로 낮음	상대적으로 높음



6. 반복문

6-1. while 문

while

- 특정 조건이 True일 경우 코드 블록을 반복적으로 실행하는 문법
- 주로 반복 횟수를 모를 때 사용.
- 조건이 False 일 때 까지 계속해서 코드 실행.

while 기본 구조

```
while 조건식 :
    True 일 동안 실행될 코드
```

예제

- **while 예제1**

```
kg =80

while kg < 100 :
    print(”살이 1kg 쪘다”)
    print(f“아직 {kg}kg이네.”)
    kg += 1

print(“망했다 100kg다”)
```

->#실행 결과 :

```
살이 1kg 쪘다.
아직 80kg이네.
살이 1kg 쪘다.
아직 81kg이네.
살이 1kg 쪘다.
아직 82kg이네.
...
...
살이 1kg 쪘다.
아직 99kg이네.
망했다
조건이 False될 때까지
진행.
```

- **while 예제2**

```
money = 0
while True
    print(f”현재 내 돈은 {money}.)
    money += 100
    print(”Show me the money”)
```

->#실행 결과 :

```
현재 내 돈은 0.
Show me the money
현재 내 돈은 100.
Show me the money
현재 내 돈은 100.
Show me the money
...
현재 내 돈은 67100.
Show me the money
...
무한 증식
while 조건이 True면 무한 루프.
```

6. 반복문

dk techin
a kakao company

6-2. for 문

for

- 특정 조건이 True일 경우 코드 블록을 반복적으로 실행하는 문법
- 주로 반복 횟수를 모를 때 사용.
- 조건이 False 일 때 까지 계속해서 코드 실행.

for 기본 구조

```
for 변수 in 반복 가능한 객체 :
    반복 구문
```

예제

- **for 예제1**

```
fruits = [“사과”, “바나나”, “복숭아”, “키위”]

for fruit in fruits :
    print(fruit)

->#실행 결과 :

사과
바나나
복숭아
키위
```

for문은 리스트의 각 항목을 순차적으로
변수에 대입하여 print() 함수를 통해 출력

- **for 예제2**

```
number = [1, 2, 3, 4, 5]
plus = 0

for num in number :
    plus += num

print(plus)

->#실행 결과 : 15
```

모든 원소를 더하는 코드
number의 항목을 순차적으로 가져와 num에 저장하며
저장 할 때마다 plus에 그 값을 합침.

6. 반복문

dktechin
a kakao company

6-2. for 문

for 변수 in range(횟수)
반복구문

- for 변수 in range(stop)
- for 변수 in range(start, stop)
- for 변수 in range(start, stop, step)

■ 예제

- **for 변수 in range(stop) 예제**

```
for x in range(6) :
   print(x)
```

```
->#실행 결과 :
1
2
3
4
5
```

#0 부터 stop-1 까지 숫자를 생성
0 부터 stop = 6-1 = 5까지 순차적으로 출력

- **for 변수 in range(start, stop) 예제**

```
for i in range(1, 10)
 print( 6 * i )
```

```
->#실행 결과 :
6
12
18
24
30
36
42
48
54
```

# start 부터 stop-1 까지 숫자를 생성
start = 1 부터 stop = 10 – 1 = 9 까지
순차적으로 출력

- **for 변수 in range(start, stop, step) 예제**

```
for i in range ( 2, 10, 2)
   print(i)
```

```
->#실행 결과 :
2
4
6
8
10
12
14
16
18
```

# start 부터 stop-1 까지 step 간격으로 생성
start = 2 부터 stop = 10 – 1 = 9 까지 step = 2
간격으로 순차적으로 출력

6. 반복문

dktechin
a kakao company

6-2. for 문

for 변수 in enumerate()
반복구문

- 순서가 있는 자료형을 입력 받을 시 인덱스와 값을 포함하여 출력
- 각 요소의 인덱스와 값을 튜플 형태로 함께 반환하는 내장 함수로 for문과 주로 쓰임
- 각 요소를 가져올 때 index를 안 붙여도 된다는 장점이 있어서 매우 편함.

예제

- **for 변수 in enumerate() 예제1**

```
fruits = [“apple”, “banana”, “cherry”]

for fruit in enumerate(fruits) :
    print(fruit)
```

```
->#실행 결과 :
0, apple
1, banana
2, cherry
```

#인덱스와 값을 튜플 형태로 함께 반환

- **for 변수 in enumerate(자료형데이터, start=?) 예제2**

```
fruits = [“apple”, “banana”, “cherry”]

for index, fruit in enumerate(fruits, start=2 ) :
    print(index, fruit)
```

```
->#실행 결과 :
2, apple
3, banana
4, cherry
```

#인덱스와 값을 위와 같이 변경 가능.

6. 반복문

6-2. 리스트 컴프리헨션

리스트 컴프리헨션

- 파이썬에서 리스트를 생성하는 매우 간결하고 효율적인 방법.
- for문과 조건문을 한 줄에 표현하여, 기존에 여러 줄로 작성하던 코드를 간소화 가능

리스트 컴프리헨션 기본구조

표현식 for 변수 in 반복 가능한 객체 if 조건:

키워드	설명
표현식	각 요소에 연산이나 변환
변수	반복 가능한 객체의 각 요소를 나타내는 임시변수
반복 가능한 객체	반복 가능한 모든 데이터 객체 사용 가능
if 조건	조건을 만족하는 요소만 선택 필수 X

예제

- **간단한 예제1**

```
even_numbers = [num for num in range(10) if num % 2 ==0]
print(even_numbers)
```

->#실행 결과 : [0, 2, 4, 6, 8]
위와 같은 한 문장으로 0에서 9까지 짝수 출력 가능.

- **간단한 예제2**

```
squares = [ x**2 for x in range(1, 11)]
print(squares)
```

->#실행 결과 : [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
위와 같은 한 문장으로 1에서 10까지 숫자 제곱 출력 가능.

7. 함수

7. 함수

세부 목차

7. 함수

7-1. 기본 구조 및 키워드

함수

- 반복되는 부분이 있을 경우 한 뭉치로 묶어 모듈화 함으로써 재사용 성을 높임.
- 작성한 프로그램의 기능 단위의 함수로 분리 해놓으면 일목요연하게 파악 가능.
- 프로그램의 흐름을 잘 파악할 수 있고 오류 검출도 쉬우며, 코드 가독성도 향상됨.

■ 함수 기본 구조

```
def 함수_이름(매개변수) :
    수행할 문장1
    수행할 문장2
    ...
    return 반환 값
```

키워드	설명
def	함수를 정의하는 키워드.
함수 이름	함수를 식별하기 위한 이름.
매개변수	함수에 입력으로 전달된 값을 받는 변수 없을 수도 있고 여러개일수도 있음.
인수	함수에 입력으로 간주되는 값(객체).
return	함수의 결과 값을 반환. return 문장 생략시 None이 반환.

7. 함수

7-2. 예제

dktechin
a kakao company

■ **함수 예시**

1. 함수를 이용해 평균 구하는 코드 작성

```
def average(numbers):
    total = sum(numbers) # numbers 라는 List의 값을 모두 더하는 함수
    count = len(numbers) # len은 numbers의 길이, List안의 개수를
    result = total / count    세어주는 함수
    return result

#본문
my, numbers1 = [1, 2, 3, 4, 5]
my_numbers2 =  [1, 2, 3 ]

result1 = average(my_numbers1)
result2 = average(my_numbers2)
print(result1)
print(result2)

# 실행 결과 값
3.0
2.0
```

함수 코드 한개를 이용해 두가지 결과값을 내는데 사용.

2. 함수 사용 안하고 평균 구하는 코드 작성.

```
#본문
numbers = [1, 2, 3, 4, 5]
total = sum(numbers)
count = len(numbers)
result = total / count
print(result)

numbers2 = [1, 2, 3, 4, 5]
total2 = sum(numbers2)
count2 = len(numbers2)
result2 = total2 / count2
print(result2)


# 실행 결과 값
3.0
2.0
```

각자 코드를 작성하여서 결과 값 출력.

7. 함수

7-3. 람다

lambda

- 람다 함수는 파이썬에서 한 줄로 간결하게 함수를 정의하는 방법.
- 복잡한 로직보다는 간단한 연산이나 특정 조건을 판단하는데 사용.

기본 표현식	lambda 매개변수1, 매개변수2, ... : 리턴할 반환값
장점	코드를 간결하고 쉽게 만들 수 있다. 함수를 인자로 전달하거나 임시적인 함수 만들 때 유용 고차 함수와 함께 사용하여 다양한 기능을 구현 가능
단점	복잡한 로직을 표현하기에는 한계가 있음. 가독성 떨어지며, 디버깅이 어려울 수 있음.

- **기존 함수 코드 작성 예시**

```
def add(x, y):
    return x + y

# 함수 호출
result = add(3, 4)
print(result)

->#실행 결과 : 7
```

- **람다 함수 코드 작성 예시**

```
d
add = lambda x, y: x + y

# 함수 호출
result = add(3, 4)
print(result)

->#실행 결과 : 7
```

8. 모듈

8. 모듈

세부 목차

dktechin
a kakao company

모듈

- Math 모듈
- Random모듈
- Time 모듈
- Datetime 모듈
- OS모듈
- Subprocess 모듈
- File 모듈

8. 모듈

dktechin
a kakao company

8. 표기법

모듈

- 상수, 변수, 함수, 클래스를 포함하는 코드가 집합된 파일.
- 모듈은 다른 Python 프로그램에서 불러와 사용할 수 있게 만든 코드 단위 파일.
- 모듈을 사용함으로써 코드 재 사용성 햐상, 프로그램 구조 관리가 용이함.
- 정의된 함수, 클래스, 변수를 현재 모듈로 가져오기위해서 import로 호출.

모듈 사용법

import 모듈

모듈명	설명
Math	수학에 관련된 기능
Random	무작위 값 생성할 때 사용되는 기능
Datetime	날짜 및 시간 처리 관련된 기능.
Time	무작위 값 생성할 때 사용되는 기능
Os	운영체제 관련된 기능
Subprocess	외부 프로세스 실행 관련 모듈.
File	데이터 저장이나 불러오기 관련 모듈

8. 모듈

8-1. Math, Random 모듈

math예제

import math

1. ceil 함수
print(math.ceil(3.3)) ->#결과 4
 ->올림 함수로 소수점을 올려 정수 출력

2. floor 함수
print(math.floor(2.7)) ->#결과 2
 ->내림 함수로 소수점을 내려 정수 출력

3. fabs 함수
print(math.fabs(-80)) ->#결과 80
 ->절대값 함수로 절대 값 출력.

random 예제

import random

1. random 함수
print(random.random()) ->#결과 0.314792212
->0~1사이의 무작위 실수 반환

2. uniform 함수
print(ramdom.uniform(1, 7)) ->#결과 6.71023
->2개의 숫자 사이의 무작위 실수 반환

3. randint 함수
print(random.randint(1, 6)) ->#결과 5
->2개의 숫자 사이의 무작위 정수 반환

4. randrange 함수
print(random.randint(1, 11, 3)) ->결과 7
(start,stop,step) 1부터 10사이 숫자 중 3의 간격
숫자중 1개 반환(1, 4, 7, 10)

import random
a = [1, 2, 3, 4, 5, 6]

5. choice 함수
print(random.choice(a)) ->#결과 3
-> 리스트내에서 값을 무작위 선택.

6. choices 함수
print(random.choices(a, k=3)) ->#결과 2, 2, 6
-> 리스트내에서 k개의 값을 무작위 선택 .
 (중복O)
2. sample 함수
print(ramdom.samle(a, 2)) ->#결과 1, 4
-> 리스트내에서 2개의 값을 무작위 선택.
 (중복X)
3. shuffle 함수
random.shuffle(a)
print(a) ->#결과 a=[5, 1, 2, 6, 4, 3]
원소의 순서를 랜덤하게 변경.

8. 모듈

8-2. Time, DateTime모듈

코드	설명	예시
%a	요일 줄임말	Sun, Mon, ...
%A	요일	Sunday, ...
%d	일	01, 02, ... 31
%b	월 줄임말	Jan, Feb, ... Dec
%B	월	January, ...
%m	월 숫자	01, 02, ... 12
%y	두자릿 수 연도	01, 02, ... 24
%Y	네자릿 수 연도	1991, ... 2024
%p	AM, PM	AM, PM
%I	시간(12시간)	01, 02, ... 12
%H	시간(24시간)	01, 02, ... 24
%M	분	00, 01, ... 24
%S	초	01, 02, ... 12

Time 예시

```
import time

1.  time 함수

print(time.time())     ->#결과  1718933403.120192
->Epoch Time -1970년 1월 1일 0시 0분 0초 기준
Epoch Time부터현재까지 경과된 시간을 초로 환산

2. sleep 함수

print("1분만 있다가")    ->#결과  1분만 있다가
time.sleep(60)              ->(60초 대기)
print("1분 끝")             ->  1분 끝
->주어진 시간(초)동안 실행을 일시정지.


4.ctime 함수

print(time.ctime())       ->#결과  Mon Jan 11 21:39:00 2024
print(time.ctime(50))   - >#결과  Thu Jan 1 09:00:50 1970
->인자 없이 사용시 현재 시간 출력
->인자 사용시 Epoch Time 기준으로 초단위로 경과된
   시간 값 반환.
```

Datetime 예시

```
import datetime

1.  datetime 함수

dt = datetime.now()

print(dt)          -> 2024,12,10,17,28,30, 518202
print(dt.year)          -> 2024
print(dt.month)         -> 12
print(dt.day)           -> 10
print(dt.hour)          -> 17
print(dt.second)        -> 30
print(dt.microsecond)   -> 518202
 -> now()는 컴퓨터의 현재 시간을 datetime을
클래스 객체로 만들어 반환

2. strftime, strptime 함수

datetime.strftime( dt, '%Y-%m-%d' )
 ->datetime을 문자열로 반환

datetime.strptime( dt, '%Y-%m-%d %H:%M:%S' )
 ->문자열을 datetime로 반환.
```

8. 모듈

8-3. OS 모듈

OS 모듈

- OS 모듈은 Python에서 운영체제와 상호작용하기 위해 다양한 기능을 제공.
- 파일시스템 관리, 디렉토리 생성 및 삭제, 파일 읽기, 쓰기 등 운영체제 관련 작업 수행 시 사용.

OS 모듈 예시

- **import os**

- os.chdir('test/')
-> 현재 작업 경로 변환

- os.remove("folder.txt")
-> folder.txt 파일 삭제.

- os.listdir()
-> 현재 디렉토리에 있는 파일과 디렉토리 목록 반환

- **import os**

- os.mkdir("myfolder")
-> myfolder 디렉토리 생성

- os.rmdir("myfolder")
-> myfolder 디렉토리 삭제

- os.getcwd()
-> 현재 작업 디렉토리 경로를 반환.

- **import os**

- os.path.isfile("myfolder")
-> myfolder 파일 존재 확인

- os.path.isdir("~/test")
-> ~/test 디렉토리 존재 확인.

- os.path.exists("~/test")
-> 파일 또는 디렉토리 존재 확인.

- os.path.join('~test/, ~/test2')
-> 두개의 경로를 합침.

- os.path.split('myfolder')
->경로를 디렉토리와 파일 이름으로 분리.

8. 모듈

8-3. OS 모듈

os.system

- 파이썬 코드 내에서 셸 명령어를 직접 실행하는 가장 간단한 방법.
- 마치 터미널에서 명령어를 입력한 것과 같은 효과

■ os.system() 활용해서 파티션 + 파일시스템 생성 스크립트 만들기

```
import os

#파티셔닝 하는 함수
def create_partition(device):
    os.system(f"parted –s /dev/{device} mklabel gpt")
    os.system(f"parted –s /dev/{device} mkpart parmary 2048s 100%")

#파일시스템 만드는 함수
def create_file_system(device, flie_system_type):
    os.system(f"mkfs.{file_system_type} –f /dev{device}1"
```

올바른 버전

```
def.main():
    device = input("디바이스를 입력. ex) sdb")
    file_system_type = input("파일 시스템 타입을 입력. ex) xfs")

    create_create_partition(device)
    create_file_system(device)

if _name_ == "_mail_":
    main()
```

잘못된 버전

```
def.main():
    device = input()
    create_create_partition(device)
    create_file_system(device)

if _name_ == "_mail_":
    main()
->위 예제는 sudo와 올바른 실행파일 경로를 안붙였으며
    mkfs, parted 실행 경로가 올바르지 않음.
```

참고 사항

1. os.system(), 앞으로 배울 os기능, subprocess모듈을 활용 시 쉘 명령어를 실행 불가 경우 실행 파일 경로를 적어야 함.
2. 콘솔창이나 터미널에서 실행해야하는 프로그램들은 /usr/bin or /usr/sbin or /bin or /sbin 에 위치하며 실행 파일 위치 확인 후 경로 작성.
3. 명령어 앞에 sudo 작성 필수

8. 모듈

8-3. OS 모듈

os.popen()	• 셸 명령어를 실행하고, 그 결과를 파일처럼 읽을 수 있도록 하는 기능을 제공 • 파일에 대한 객체를 읽기 위해서 read(), readline() 함수를 사용해야 print 시 정보를 읽어 올 수 있음.(단 readline으로 읽을때, 배열로 들어옴)

os.popen

- **기본 형태**

```
import os

pipe = os.popen("명령어")
output = pipe.read()
pipe.close()
```

- **ls-al명령어 압력 후 읽어보기**

```
import os

def main()
   out = os.popen("sudo ls –al")
   print(out.read())

if _name_ == " _main_" :
   main()
```

```
//실제 출력
total 120
drwxr-xr-x+   44  jeongmingi  staff    1408  Jun 19 21:59  .
drwxr-xr-x     6  root         admin     192  Jan 17  2022  ..
-r--------     1  jeongmingi  staff       8  Jan 21  2020 .CFUserTextEncoding
-rw-r—r--@     1  jeongmingi  staff   14340 Aug  4  14:23 .DS_Store
....
```

8. 모듈

8-4. subprocess 모듈

dktechin
a kakao company

subprocess

- 외부 프로그램을 실행하고, 그 실행 결과를 제어하는 도구.
- 셸 명령어를 실행하거나, 다른 프로그램 출력을 Python 코드에서 처리 등 다양하게 작업에 활용.
- OS 모듈에 비해 표준 입출력 제어하는 등 더 다양한 기능 제공.
- subprocess.run(), 함수와 제일 중요한 subprocess.Popen() 함수에 대해 알아보자.

8. 모듈

dktechin
a kakao company

8-4. subprocess.run()

Subprocess.run() 기본 형태

```
import subprocess
result = subprocess.run(arg, *, capture_output=False, text=False, Shell=False, cwd=None, timeout=None, input=None, env=None, universal_newlines=None)
```

매개변수	설명
args	실행할 명령어를 리스트 형태로 전달(예:[“ls”,”-la”])
captue_output	명령어의 표준 출력과 표준 에러를 Python 변수로 저장할지 여부를 설정
text	출력 결과를 문자열 형태로 반환할지 여부를 설정
check	명령 실행 결과가 0이 아니면 subprocess.CalledProcessError 예외를 발생.
shell	셸을 사용하여 명령을 실행할지 여부를 설정
cwd	작업 디렉토리를 설정
timeout	명령 실행 시간 제한을 설정
input	프세스의 표준 입력으로 전달할 데이터
env	프로세스 실행 환경 변수를 설정

Subprocess.run() 활용하여 Hello 출력 예제

```
import subprocess

def main()
    cmd = “echo Hello”.split(" ")      #[”echo”, “Hello”]
    subprocess.run(cmd)

if _name_ == “ _main_” :
    main()

#출력
#Hello
```

- 문자열 split 함수를 활용하여 띄어쓰기 단위로
- 리스트 만들어 줌.

8. 모듈

- 8-4. subprocess.run()

Subprocess.Popen() 정의

- **subprocess.Popen() 함수는 파이썬에서 외부 프로세스를 생성하고 관리하는데 사용되는 도구.**
- **os.popen()과 달리 더욱 세밀한 제어가 가능하며, 다양한 상황에 맞춰 사용할 수 있는 많은 매개 변수를 제공.**

매개변수	설명
args	실행할 명령어와 인수들을 리스트 형태로 전달 (예:[“ls”,”-la”])
stdin, stdout, stderr	프로세스의 표준 입력, 출력, 에러 파이프 설정
cwd	프로세스의 작업 디렉토리 설정
env	프로세스의 환경 변수를 설정
shell	True 로 설정하면 셸을 통해 명령을 실행
universal_newlines	True로 설정하면 줄 바꿈 문자를 자동 변환
start_new_session	True로 설정하면 새로운 세션에서 프로세스 시작
preexec_fn	프로세스 실행 전에 호출할 함수를 지정
close_fds	True로 설정하면 모든 파일 디스크립터를 닫음

메소드	설명
communicate (input=None, timeout=None)	프로세스와 통신하고, 입력을 전달하고 출력을 받음
wait(timeout=None)	프로세스가 종료될 때까지 기다림
poll()	프로세스의 상태를 확인.
terminate()	프로세스를 종료함
kill()	프로세스를 강제 종료함

Popen 생성자를 이용해서 만든 객체가 제공하는 메소드

8. 모듈

8-4. subprocess.Popen()

Subprocess.Popen() 활용 예제

```
import subprocess


#현재 디렉토리 목록 출력
process = subprocess.Popen([“ls”, “-al”], stdout=subprocess.PIPE)
output, error = process.communicate()
print(out.decode(‘utf-8’)
```

전체 내용 요약

- subprocess 모듈을 이용하여 ls –al 명령을 실행
- 실행결과(현재 디렉토리 목록)을 파이썬 변수 output에 저장
- 저장된 결과를 UTF-8 인코딩으로 디코딩 후 변환 후 출력
- 즉, 현재 디렉토리의 상세 목록을 화면에 출력하는 코드

1번 코드줄 살펴보기

➢ process = subprocess.Popen([“ls”, “-al”], stdout=subprocess.PIPE)

- subprocess.Popen(): 새로운 프로세스를 생성하는 함수.
- [“ls”, “-al”]: 실행할 명령어와 인수를 리스트 형태로 전달함 이 경우에 현재 디렉토리의 상세 목록을 출력하는 ls –al 명령을 실행함. (명령어 전달할 때는 리스트의 형태로 전달함.)
- stdout=subprocess.PIPE: 프로세스의 표준 출력을 파이프로 연결하여 파이썬 프로그램에서 읽을 수 있도록 설정.

2번 코드 줄 살펴보기

➢ output, error = process.communicate()

- process.coumunicate() : 생성된 프로세스와 통신하여 출력을 받는 메소드
- output : 프로세스의 표준 출력을 저장.
- error : 프로세스의표준 에러를 저장.

3번 코드 줄 살펴보기

➢ output, error = process.comunicate()

➢ out = proc.communicate()[0]
➢ err = proc.communicate()[1] -> 위 코드와 같은 코드

4번 코드 줄 살펴보기

➢ print(out.decode(‘utf-8’)

- 위 output은 바이트열 형태로 문자열로 변환하기 위해 decode() 메소드를 사용

8. 모듈

dk techin
a kakao company

8-4. subprocess.Popen()

Subprocess.Popen() 파이프라인 연결 방법 및 문제점

```
import subprocess

process = subprocess.Popen([“ls”, “-al”, “|”, ”grep”, “python”], stdout=subprocess.PIPE)
output, error = process.communicate()
print(out.decode(‘utf-8’))
```

```
# subprocess 사용하여 ‘ls -al |grep python’ 을 출력하기 위해  보통 위 코드처럼 작성하지만, 위 코드 는 에러 발생 및 실행이 안됨.
#오류를 발생 시키지 않기 위해선 달느 프로세스의 출력을 stdin으로 입력 받아,매개변수를 입력하여 해결하여야 함.
import subprocess
```

```
process1 = subprocess.Popen([“ls”, ”-al”], stdout=subprocess.PIPE)
process2 = subprocess.Popen([“grep”, “python”], std=process.stdout, stdout=subprocess.PIPE)   #여기 부분

output, error = process2.comunicate()
print(out.decode(‘utf=8’))

#process1 : ls –al 명령 실행.
#procees2 : grep 명령 실행 (process1의 출력을 입력으로 받음)
```

- 결론

subprocess.communicate()는 파이썬에서 외부 프로세스를 제어하고 데이터를 주고 받는 강력한 도구로
파이프 라인을 구성하여 복잡한 작업을 수행 할 수 있으며, 다양한 시스템 관리 작업에 활용 가능.

8. 모듈

8-5. file 모듈

File 모듈

- python에서 파일을 다루는 것은 중요한 작업으로 데이터를 저장하거나 불러오기 위해 사용.
- 다양한 형식의 파일을 처리 할 수 있습니다.
- file 객체는 따로 import 안해줘도 사용가능
- 파일을 열때 open() 함수 사용하고 작업이 완료되면 close() 함수로 닫아주는 것이 일반적.

File 모듈 기본 형태

file_obj = open(파일명, mode)

mode	설명
"r"	읽기 모드(기본)
"w"	쓰기 모드(기존 파일 내용 삭제)
"a"	추가 모드(파일 끝에 내용 추가)
"x"	파일이 존재하지 않을 때만 생성

파일 읽기	설명
read()	파일 전체를 한 번에 읽어 문자열로 반환
readline()	파일을 한 줄씩 읽어 문자열로 반환.
readlines()	파일 전체를 읽어 각 줄을 요소로 하는 리스트로 반환.

8. 모듈

8-5. File 모듈

File모듈 예시

```
f = open(“my_file.xtx”, “r”)   #파일 열기 (읽기 모드)

for line in f:                 # 파일 내용 읽기 ( 예시: 한 줄씩 읽기)
   print(line, end=“”)

f.close                        #파일 닫기
```

File모듈로 txt 파일 만들기

```
fw = open(“new.txt”, ‘w’)

for x in range(3)
   fw.write(“안녕하세요” \n”)

fw.close
```

```
#실행 결과
안녕하세요
안녕하세요
안녕하세요
```

- 먼저 new.txt 파일이 있다면 덮어쓰거나새로운 new.txt를 만듭니다.
- 이 후 for문을 이용하여 안녕하세요 문구를 3번 입력합니다.
- 이 후 파일 객체를 닫습니다

with as 문 사용 예시

```
with open(“new.txt”, ’w’) as fw:
   for x in range(3)
      fw.write(“ 안녕하세요”)
```

```
#실행 결과
안녕하세요
안녕하세요
안녕하세요
```

- with, as를 사용하면 file 모듈을 좀 더 간결히 작성 가능.
- with, as문을 사용하면 with는 파일 객체를 생성하고 블록을 빠져나갈때 자동으로 파일을 닫음. as fw 부분은 fw라는 별칭으로 파일 객체를 참조.

- 결론

with 문은 python에서 파일을 안전하고 효율적으로 다루는 데에
필수적인 문법으로 작업 시 with as문 사용을 권장.

9. 네이밍 작성법

9. 네이밍 작성법

세부 목차

snake_case

- 표기법
- 변수 및 함수 네이밍

9. 네이밍 작성법

dktechin
a kakao company

9-1. 표기법

표기법	• 변수나 함수 같이 이름을 작성 시 snack_case를 사용하여 작성 • 네이밍을 통해 코드의 가독성을 높이고 유지 보수에 용의하게 함. • 변수 네이밍엔 최대한 자세하게 하되 불필요한 이름은 짓지 말 것. 전치사는 필수X

Snake_case 란?

- 모든 단어가 소문자 or 대문자 시작하고, 단어와 단어 사에는 언더바(“_”)로 연결된 방식을 뜻한다.
- 뱀의 형태를 지녀서 snake_case 라고 불리우며 스네이크 표기법이라고도 한다.
- 예를 들어 user의 id를 네이밍 할 경우 언더바(“_”)를 활용하여 이름을 짓게 되면 user_id 라는 변수 이름이 나오게 된다.
- 전반적으로 Python이나 C언어는 보통 snack_case라는 표기법을 사용한다.
- 그 외로 java는 Camel Case 낙타표기법, C#은 Pascal Case 파스칼 표기법을 사용하며 각 언어마다 표기법이 있으니 참고 요망

9. 네이밍 작성법

dktechin
a kakao company

9-1. 변수 및 함수 네이밍

변수 네이밍

1. 단수 인 경우 – 언더바(“_”) 사용.

```
user_name
```

2. 리스트나 튜플 – 끝에 s의 복수 형태 or list와 같은 네이밍 사용.

```
student_grades = [ 90, 80, 300 ]
student_grade_list = [90, 80, 300]
```

3. 딕셔너리 – key와 value로 이루어져 포괄적인 의미를 사용.

```
user = { “name” : Alice”, “age” : 30, “hobby” : “game” }
```

4. 상수 – 대문자와 언더바(“_”) 사용.

```
PI= 3.14159
RED_COLOR = (255, 0, 0)
GREEN_COLOR = (0, 255, 0)
BLUE_COLOR = (0, 0, 255)
```

함수 네이밍

1. 함수 네이밍 – 함수는 보통 동사나 명사 형태.

예시) IMS API로 부터, 서버의 정보를 가져온다면 다음과 같음.

```
def getServerFromImsAPI():
    result = ...
    return result
```

2. 함수의 리턴값이 True, False를 반환한다면 can, has, be동사+명사로 네이밍

```
def is_valid()
    ....
    return True

def can_usre_id():
    ....
    return False
```

10. 코딩 연습

10. 코딩 연습

10-1. 코딩 연습

문법익히기

- 개인적으로 처음 코딩을 연습하시면, 문법 연하기에는 백준 알고리즘이나, 프로그래머스를 추천합니다.

백준알고리즘

- https://solved.ac/problems/level 에서 브론즈 3~5 정도 연습을 추천드립니다.

프로그래머스

- https://programmers.co.kr/ 에서 코딩 테스트 입문 -> 정답률 높은 문제로 카테고리 설정 후 문제 풀이 연습을 추천드립니다.

참고

- 문제 풀이시 막히시면, 구글에 검색하면 사람들이 올려 놓은 코드를 참고하셔서 풀이하시길 권장드립니다.
- 검색어 ex) Python 프로그래머스 몫 구하기

dk techin
a kakao company

감사합니다.

http://www.dktechin.com