TypeError

ValueError

파이썬 인터프리터 -> ***훨씬 더 사용하기 쉽고 운영체제간 이식도 가능 (확장성)***

표현식 -> evaluation

표현식이 \****평가***되어 값이 반환됨

***\*표현식이나 문장을 순차적으로 평가하여 프로그램의 동작을 결정***

float는 실수에 대한 ***근삿값***

이로 인해 생기는 오류***floating number round error***

시퀀스 타입 : 여러 개의 값들을 ***순서대로 나열***하여 저장하는 자료형

튜플은 ***개발자가 직접 사용하기 보다 ‘파이썬 내부 동작’에서 주로 사용됨***

함수 쓰면 -> ***재사용성***, 코드의 ***가독성과 유지보수성*** 향상

***위치인자는 함수 호출 시 반드시 값을 전달해야 함***

***키워드인자는 호출 시 반드시 위치 인자 뒤에 있어야 함***

에러메세지 : positional argument follows keyword argument

임의의 인자 목록 = \*param -> tuple로 처리 / 임의의 키워드 인자 목록 \*\*param -> dict로 처리

함수 인자 권장 작성 순서

위치 -> 기본 -> 가변 -> 가변 키워드

***단, 모든 상황에 적용되는 절대적인 규칙은 아니며,***

***상황에 따라 유연하게 조정될 수 있음***

del sum -> sum 변수 객체 삭제 (sum() 함수 쓰고싶은데 앞에서 sum 변수 만들었으면 에러 뜨므로 객체를 삭제 해줘야함)

함수는 코드 내부는 local scope, 그 외는 global scope로 구분함

***변수 수명주기(lifecycle)***

1. built-in scope : 파이썬이 실행된 이후부터 영원히 유지
2. global scope : 함수가 호출된 시점 이후 혹은 인터프리터가 끝날 때까지 유지
3. local scope : 함수가 호출될 때 생성되고, 함수가 종료될 때까지 유지

이름 검색 규칙 - LEGB (Local, Enclosed, Global, Built-in) Rule

***함수 내에서는 바깥 Scope의 변수에 접근 가능하나 수정은 할 수 없음***

global 키워드는 가급적 사용 X

함수로 값을 바꾸고자 한다면 항상 ***인자***로 넘기고 함수의 ***반환 값***을 사용하는 것을 권장

함수 내 글로벌 변수 선언 시

1. 선언 전에 해당 변수에 접근하면 에러

2. 매개변수와 같은 이름으로는 선언 불가

재귀함수 : 함수 내에서 자신을 호출하는 함수 (ex. 팩토리얼)

base case 필수 (종료조건 명확히하고 재귀가 그쪽으로 향하도록 할 것)

유용한 내장함수

* map(function, iterable), zip(\*iterable) -> 둘 다 오브젝트 반환함
* lambda 매개변수: 표현식

패킹

* packed\_values = 1, 2, 3, 4, 5, 변수1, 변수2, 등등 => 튜플로 묶음
* a, \*b, c = [1, 2, 3, 4, 5]
* print(\*object, sep=’ ‘, end=’\n’, file=sys.stdout, flush=False)

언패킹

* a, b, c, d, e = (iterable)
* print(\*b) => 출력 : 2 3 4
* 함수(\*\*딕셔너리) => 함수가 딕셔너리의 키-값 쌍을 키워드 인자(키=값)로 받음

<모듈>

***. 은 왼쪽 객체에서 오른쪽 이름을 찾아라 라는 의미 ex. 모듈.변수, 모듈.함수()***

***from*** 모듈 import 요소(변수나 함수)

모듈 주의사항 : ***만약 서로 다른 모듈이 같은 이름의 함수를 제공할 경우*** 마지막으로 import된 이름으로 대체됨

사용자 정의 모듈 : 모듈 파일 (.py) 작성 -> 요소 작성, 모듈 import 후 요소 호출

파이썬 표준 라이브러리(PSL) : 파이썬에 기본 제공되는 다양한 모듈과 패키지 모음

패키지 : 모듈들을 하나의 디렉토리에 모은 것

**from** directory.module **import** 요소

외부 패키지 설치 -> pip install (패키지 이름)

pypi.org에 저장된 패키지 설치

<제어문, 반복문>

제어문 : ***조건***에 따라 코드 블록을 실행하거나 ***반복***적으로 코드를 실행

for : 반복 횟수가 명확하 경우 유용

while : 반복 횟수가 불명확한 경우, 조건에 따라 반복을 종료해야하는 경우 유용

for i in outers:

for j in inners:

print(i, j)

-> print가 호출되는 횟수 = ***len(outers) \* len(inners)***

while문은 반드시 ***종료 조건***이 필요

continue : ***현재 반복문의 남은 코드를 건너뛰고 다음 반복으로 넘어감***

break와 continue를 남용하는 것은 가독성을 저하시킬 수 있으므로

***특정한 종료 조건***을 만들어 break를 대신하거나, ***if 문을 사용***해 continue 처럼 건너 뛰면 좋음

enumerate(iterable, start=0) : 각 요소와 그 인덱스를 반환하는 내장함수

\n

[start. stop, step]

str은 원소 변경 불가능

리스트 다루는 거 시험에 나옴~ (추가 삭제 뭐 등등)

Sequence type의 특징 -> 순서가 있음

리스트 = [ ]

가변적

range를 쓰는 이유 : 메모리 사용을 줄이기 위해 (리스트로 레인지처럼 만드는 것 보다)

range가 리스트처럼 보이지만 실제로 리스트를 만들지 않아 공간을 절약한다.

튜플 = ( )

튜플의 특이한 점 : 요소가 하나 일 때 (1) = 1 으로 인식돼서 튜플 = (1,) 이렇게 써야함

* 불가변적
* 대부분은 개발자가 의도적으로 쓰진 않고 x, y = 1, 2 처럼 쓰면 컴퓨터는 x, y = (1, 2) 로 처리함

인덱스 접근 -> name[index] / name[start:stop:step]

non-sequence

dict = { a : b }

* 순서x, 중복x
* key(불가변) - value(가변) 쌍
* 튜플이나 리스트처럼 순서(index)로 접근하지 못하고 key로만 value에 접근 및 수정 가능
  + name[‘key’]
* 마지막에 넣은 key만 적용됨 (중복key가 허용되지 않기 때문)
* 그래프 풀 때 유용함

set = { }

* 순서, 중복 X
* 가변적
* 수학에서의 집합과 동일한 연산처리 가능
* set1 | set2 합집합, set1 - set2 차집합, set1 & set2 교집합

None

* 어떠한 값이자 타입 이름
* a = None

boolean

* True, False -> 비교/논리 연산자와 조건/반복문에 사용

collection

|  | 가변성 | 순서 |
| --- | --- | --- |
| str | X | O sequence |
| list | O | O |
| tuple | X | O |
| set | O | X non-sequence |
| dict | O | X |

암시적 형변환

* int + float = float
* True + 1 = 4
* True + False = 1

명시적 형변환 (개발자가 직접 바꿔줘야하는 경우)

* str -> int : 형식에 맞는 숫자만 가능
* int -> str : 모두 가능

산술 연산자

* 복합 연산자 : += 같은 것들

비교 연산자

* == : 값만 비교 / is : 주소 비교 (2.0 is 2 = False)
* is 는 되도록이면 None T/F 등을 비교할 때 사용

논리 연산자

* and, or, not ~
* 단축평가 : 논리 연산에서 두 번째 피연산자를 평가하지 않고 결과를 결정하는 동작
  + Falser and ~ : 이미 False 이므로 단축평가가 시행됨 (T or ~ 도 마찬가지)

vowels = ‘aeiou’

print((‘a’ and ‘b’) in vowels) # False

print((‘b’ and ‘a’) in vowels) # True

print(3 and 5) # 5

print(3 and 0) # 0

print(0 and 3) # 0

print(0 and 0) # 0

print(5 or 3) # 5

print 3 or 0) # 3

print(0 or 3) # 3

print(0 or 0) # 0

프린트에서의 논리연산 단축평가는 마지막으로 평가한 피연산자가 리턴됨

조건문 등에서는 T/F로 출력 됨

0인 것과 0이 아닌 것 / null과 문자

멤버십 연산자

* ~ in 시퀸스
* ~ not in 시퀸스

시퀸스형 연산잔

* + 결합 연산자
* \* 반복 연산자

1.17

<함수>

함수를 사용하는 이유

* 코드의 중복을 방지
* 재사용성, 코드 가독성, 유지보수성 향상

내장함수

파이썬이 기본적으로 제공하는 함수

별도의 import 없이 바로 사용 가능 (대표적으로 print, abs)

함수 호출

함수의 이름 → 해당 함수의 코드 블록을 실행

함수의 구조

매개변수와 인자

위치인자

기본인자값

키워드 인자

임의의 인자 목록 func(\*parameter) -> 튜플로 가변인자를 받음

임의의 키워드 인자 목록 (\*\*parameter) -> key=value 형식의 가변 키워드 인자를 dict로 묶어 처리

함수와 스코프

전역변수, 지역변수 -> 기준=함수

작은 범위에서 큰 범위로는 수정, 찾기 등 작동하지만 큰 범위에서 작은 범위로는 불가능

호출된 값 -> 지역변수 -> 전역변수 순으로 값을 찾아나가면 됨

global 변수 : 변수를 글로벌로 선언

재귀함수

큰 문제를 작은 문제로 풀어나갈 때 씀

ex 팩토리얼 : 큰 수 -> 작은 수 -> 1 (=> base case/수렴)

유용한 함수

map(func, iterable(=반복 가능한 데이터 ex. 시퀸스)

iterable의 모든 요소에 func를 적용해 오브젝트 형성

func는 내장함수 뿐만 아니라 내가 만든 함수도 가능 (lambda가 유용)

zip(\*iterables) -> 각 iterable을 튜플 원소로 모아 오브젝트 형성

list(zip(iterable1, iterable2)

lambda 함수 = 익명 함수

lambda 매개변수: 표현식(body)

def 없이 간단히 만들 때 유용.

한줄로 일회성 함수를 쓰고싶을 때 자주 씀 (map이랑 자주 쓰임)

<packing & unpacking>

패킹

여러 개의 값을 하나의 변수에 묶어서 담는 것 ex. 가변인자, print

프린트 함수의 원형

print(\*object, sep=’ ‘, end=’\n’, file=sys.stdout, flush=False)

언패킹 예시

시퀸스 각 원소를 하나하나 변수에 넣는 일

\*와 \*\*는 패킹도 하지만 언패킹도 함

names = [a,b,c]

print(\*names) # a b c

my\_dict = {‘x’ : 1, ‘y’ : 2, ‘z’ : 3}

print(\*\*my\_dict) # 1 2 3

1.18

<모듈>

모듈 모음 => 패키지 모음 => 라이브러리

파이썬 표준 라이브러리

Python Standard Library

기본 내장 된 라이브러리

pip : 외부 패키지들을 설치하게 해주는 시스템

<제어문>

반복 횟수를 정할 수 있으면 for 없으면 while

for \_ in list, tuple, str, set, dict,

dict는 처음 인자에 key, 두번째 인자에 value가 들어감 (하나만 있으면 key만 들어감)

ex) for key, value in dict

im테스트

중첩된 리스트를 다루는 실력을 평가