CHAPTER4 파이크러스트 : 코드 구조

4.1 코멘트 달기 : #

comment : 인터프리터에 의해 무시되는 텍스트의 한 부분, 코드를 설명하거나 나중에 어떤 문제를 고치기 위해 표시

4.2 라인 유지하기 : 백슬래시

한 라인에서 권장하는 최대 문자수는 80자

이 길이 안에 넣고 싶은 코드를 모두 입력할 수 없다면 백슬래시 문자를 입력한 후 다음 라인에 계속 입력

4.3 비교하기 : if, elif, else

if 와 else : 조건이 참인지 거짓인지 확인하는 파이썬의 선언문

print() : 일반적으로 화면에 객체를 출력하는 파이썬의 내장 함수

4.4 반복하기 : while

파이썬에서 가장 간단한 루핑 매커니즘

>>> count = 1

>>> while count <= 5:

... print(count)

... count += 1

... 1 2 3 4 5

>>>

4.4.1 중단하기 : break

어떤 일이 일어날 때까지 반복하고 싶지만, 어떤 일이 언제 일어날지 확실하지 않을 때 사용

>>> while True:

... stuff = input("String to capitalize [type q to quit]: ")

... if stuff == "q":

... break

... print(stuff.capitalize())

... String to capitalize [type q to quit]: test

Test

String to capitalize [type q to quit]: hey, it works

Hey, it works

String to capitalize [type q to quit]: q

>>>

4.4.2 건너뛰기 : continue

반복문을 중단하고 싶지는 않지만 몇몇 이유로 다음 루프로 건너뛰고 싶을 때 사용

>>> while True:

... value = input("Integer, please [q to quit]: ")

... if value == 'q': # quit

... break

... number = int(value)

... if number % 2 == 0: # an even number

... continue

... print(number, "squared is", number\*number)

...

Integer, please [q to quit]: 1

1 squared is 1

Integer, please [q to quit]: 2

Integer, please [q to quit]: 3

3 squared is 9

Integer, please [q to quit]: 4

Integer, please [q to quit]: 5

5 squared is 25

Integer, please [q to quit]: q

>>>

4.4.3 break 확인하기 : else

break는 어떤 것을 체크하여 그것을 발견했을 경우 종료하는 while 문을 작성할 때 사용

while문이 모두 실행되었지만 발견하지 못했을 경우에는 else가 실행됨

>>> numbers = [1, 3, 5]

>>> position = 0

>>> while position < len(numbers):

... number = numbers[position]

... if number % 2 == 0:

... print('Found even number', number)

... break

... position += 1

... else: # break not called

... print('No even number found')

... No even number found

4.5 순회하기 : for

딕셔너리의 values() : 키보다 값 순회

items() : 튜플에서 키와 값을 모두 반환

4.5.1 중단하기 : break

4.5.2 건너뛰기 : continue

4.5.3 break 확인하기 : else

4.5.4 여러 시쿼스 순회하기 : zip()

여러 시퀀스를 병렬로 순회

>>> days = ['Monday', 'Tuesday', 'Wednesday']

>>> fruits = ['banana', 'orange', 'peach']

>>> drinks = ['coffee', 'tea', 'beer']

>>> desserts = ['tiramisu', 'ice cream', 'pie', 'pudding']

>>> for day, fruit, drink, dessert in zip(days, fruits, drinks, desserts):

... print(day, ": drink", drink, "- eat", fruit, "- enjoy", dessert)

...

Monday : drink coffee - eat banana - enjoy tiramisu

Tuesday : drink tea - eat orange - enjoy ice cream

Wednesday : drink beer - eat peach - enjoy pie

4.5.5 숫자 시퀀스 생성하기 : range()

리스트나 튜플 같은 자료구조를 생성하여 저장하지 않더라도 특정 범위 내에서 숫자 스트림 반환

컴퓨터의 메모리를 전부 사용하지 않고, 프로그램의 충돌없이 아주 큰 범위 생성 가능

range(start, stop, step) 구조

* Start : 생략하면 범위는 0에서 시작
* Stop : 꼭 입력해야 하는 값
* Step : 기본값 1, -1로 지정하여 끝에서부터 거꾸로 진행할 수도 있음

4.6 comprehension

하나 이상의 이터레이터로부터 파이썬의 자료구조를 만드는 콤팩트한 방법

비교적 간편한 구문으로 반복문과 조건 테스트를 결합 가능

4.6.1 list comprehension

[ 표현식 for 항목 in 순회 가능한 객체 ]

>>> number\_list = [number for number in range(1,6)]

>>> number\_list [1, 2, 3, 4, 5]

[ 표현식 for 항목 in 순회 가능한 객체 if 조건 ]

>>> a\_list = []

>>> for number in range(1,6):

... if number % 2 == 1:

... a\_list.append(number)

...

>>> a\_list

[1, 3, 5]

4.6.2 dictionary comprehension

{ 키\_표현식 : 값\_표현식 for 표현식 in 순회 가능한 객체 }

>>> word = 'letters'

>>> letter\_counts = {letter: word.count(letter) for letter in word}

>>> letter\_counts

{'l': 1, 'e': 2, 't': 2, 'r': 1, 's': 1}

4.6.3 set comprehension

{ 표현식 for 표현식 in 순회 가능한 객체 }

>>> a\_set = {number for number in range(1,6) if number % 3 == 1}

>>> a\_set

{1, 4}

4.6.4 generator comprehension

튜플 : 컴프리헨션이 없음

>>> for number in number\_thing:

... print(number)

...

1

2

3

4

5

4.7 함수

* 정의하기
* 호출하기

argument(인자) : 함수로 전달한 값

paramenter(매개변수)

함수의 인자는 개수에 상관없이 모든 타입의 인자를 취할 수 있음

반환값도 마찬가지로 개수에 상관없이 모든 타입 반환 가능

만약 함수가 명시적으로 return을 호출하지 않으면, 호출자는 반환값으로 None을 얻음

\* None : 아무것도 없다는 뜻의 파이썬의 특별한 값

4.7.1 위치 인자

positional arguments : 값을 순서대로 상응하는 매개변수에 복사

>>> def menu(wine, entree, dessert):

... return {'wine': wine, 'entree': entree, 'dessert': dessert}

...

>>> menu('chardonnay', 'chicken', 'cake')

{'dessert': 'cake', 'wine': 'chardonnay', 'entree': 'chicken'}

4.7.2 키워드 인자

위치 인자의 혼동을 피하기 위해 매개변수에 상응하는 이름을 인자에 지정 가능

심지어 인자를 함수의 정의와 다른 순서로 지정 가능

>>> menu(entree='beef', dessert='bagel', wine='bordeaux')

{'dessert': 'bagel', 'wine': 'bordeaux', 'entree': 'beef'}

4.7.3 기본 매개변수값 지정하기

호출자가 대응하는 인자를 제공하지 않으면 기본값을 사용

4.7.4 위치 인자 모으기 : \*

함수의 매개변수에 \*를 사용할 때, \*는 매개변수에서 위치 인자 변수들을 튜플로 묶음

\*를 사용할 때 가변 인자의 이름으로 args를 사용할 필요가 없지만 관용적으로 사용

4.7.5 키워드 인자 모으기 : \*\*

키워드 인자를 딕셔너리로 묶기 위해 \*\* 사용

인자의 이름은 키고, 값은 이 키에 대응하는 딕셔너리 값

4.7.6 docstring

>>> def echo(anything):

... 'echo returns its input argument'

... return anything

길게 작성 가능, 서식 추가 가능

def print\_if\_true(thing, check):

'''

Prints the first argument if a second argument is true.

The operation is:

1. Check whether the \*second\* argument is true.

2. If it is, print the \*first\* argument.

'''

if check:

print(thing)

함수의 docstring을 출력하려면 help() 함수를 호출, 함수 인자의 리스트와 서식화된 docstring을 읽기 위해 함수 이름을 인자로 전달

>>> help(echo)

Help on function echo in module \_\_main\_\_:

echo(anything)

echo returns its input argument

4.7.7 일등 시민 : 함수

객체는 숫자, 문자열, 튜플, 리스트, 딕셔너리, 함수를 포함

일등 시민 : 함수를 변수에 할당할 수 있고, 다른 함수에서 이를 인자로 쓸 수 있으며, 함수에서 이를 반환할 수 있음

* func : 실행할 함수
* arg1 : func 함수의 첫번째 인자
* arg2 : func 함수의 두번째 인자

>>> def run\_something\_with\_args(func, arg1, arg2):

... func(arg1, arg2)

4.7.8 내부 함수

루프나 코드 중복을 피하기 위해 또 다른 함수 내에 어떤 복잡한 작업을 한 번 이상 수행할 때 유용하게 사용

4.7.9 클로져

다른 함수에 의해 동적으로 생성, 바깥 함수로부터 생성된 변수값을 변경하고 저장 가능한 함수

4.7.10 익명 함수 : lambda()

단일문으로 표현되는 익명 함수

* words : words 리스트
* func : words와 word문자열에 적용되는 함수

>>> def edit\_story(words, func):

... for word in words:

... print(func(word))

4.8 generator

파이썬의 시퀀스를 생성하는 객체

전체 시퀀스를 한 번에 메모리에 생성하고 정렬할 필요 없이, 잠재적으로 아주 큰 시퀀스를 순회가능, 이터레이터에 대한 데이터의 소스로 자주 사용됨

제너레이터를 순회할 때마다 마지막으로 호출된 항목을 기억하고 다음 값을 반환

>>> def my\_range(first=0, last=10, step=1):

... number = first

... while number < last:

... yield number

... number += step

...

>>> my\_range

<function my\_range at 0x10193e268>

>>> ranger = my\_range(1, 5)

>>> ranger

<generator object my\_range at 0x101a0a168>

>>> for x in ranger:

... print(x)

...

1

2

3

4

4.9 decorator

하나의 함수를 취해서 또 다른 함수를 반환하는 것

* \*args와 \*\*kwargs
* 내부 함수
* 함수 인자

decorator()

* 함수의 이름과 인자값을 출력
* 인자로 함수를 실행
* 결과를 출력
* 수정된 함수를 사용할 수 있도록 반환

>>> def document\_it(func):

... def new\_function(\*args, \*\*kwargs):

... print('Running function:', func.\_\_name\_\_)

... print('Positional arguments:', args)

... print('Keyword arguments:', kwargs)

... result = func(\*args, \*\*kwargs)

... print('Result:', result)

... return result

... return new\_function

4.10 namespace & scope

namespace : 특정 이름이 유일하고, 다른 네임스페이스에서의 같은 이름과 관계가 없는 것

함수 내의 지역 변수가 아닌 전역 변수를 접근하기 위해 global 키워드를 사용해서 전역 변수의 접근을 명시해야 함

함수 안에 global 키워드를 사용하지 않으면 파이썬은 로컬 네임스페이스를 사용하고 변수는 지역변수가 됨, 지역 변수는 함수를 수행한 뒤 사라짐

* locals() 함수는 로컬 네임스페이스의 내용이 담긴 딕셔너리를 반환
* globals() 함수는 글로벌 네임스페이스의 내용이 담긴 딕셔너리를 반환

4.10.1 이름에 \_와 \_\_사용

\_\_ 로 시작하고 끝나는 이름은 파이썬 내의 사용을 위해 예약되어 있음 -> 변수를 선언할 때 사용하면 안됨

4.11 에러 처리하기 : try, except

에러가 발생하도록 코드를 내버려두는 것보다, 에러가 예상되는 코드에 try문을 사용하고, 그 에러를 처리하기 위해 except 문을 사용

>>> short\_list = [1, 2, 3]

>>> position = 5

>>> try:

... short\_list[position]

... except:

... print('Need a position between 0 and', len(short\_list)-1, ' but got',

... position)

...

Need a position between 0 and 2 but got 5

try 블록 안의 코드를 실행할 때 에러가 있다면 예외가 발생하고 except 블록 내의 코드가 실행됨, 만약 try 블록 안에 에러가 없다면 except 블록을 건너뜀

인자 없는 except문을 지정하는 것은 모든 예외 타입을 잡는다는 것을 말함, 두 개 이상의 예외 타입이 발생하면 각각 별도의 예외 핸들러를 제공하는 것이 가장 좋은 방법

4.12 예외 만들기

IndexError와 같은 모든 예외는 파이썬 표준 라이브러리에 미리 정의되어 있는 것