목차

1. 기본적인 자료형과 연산자
   1. str 문자열
   2. list
   3. tuple
   4. dict
   5. set
   6. 자료형의 값을 저장하는 공간, 변수
2. 제어문
   1. if
   2. for
   3. while
3. 함수와 입출력
   1. 함수
   2. lambda
   3. 입출력
4. 추가 예제들

아이들 = IDLE(Integrated development and learning environment)

|  |
| --- |
|  |
| **1. 기본적인 자료형과 연산자** |

자료형

bool int float str list tuple dict set  
cf) 0o oct 0x hex  
cf) sequential 자료형은 인덱스로 관리 가능한 자료형을 말함.

연산자

+ - \* / // % \*\*

**1. str 문자열**

+ 연산 가능. \* 연산 가능.

**1-1. 문자열 인덱싱과 슬라이싱**

|  |
| --- |
| >>> a = "Life is too short, You need Python"  >>> a[0:4]  ‘Life’  >>> a[19:]  ‘You need Python’  >>> a[:18]  ‘Life is too short,’  >>> a[:]  "Life is too short, You need Python"  >>> a[19:-7]  ‘You need’  >>> a[-7:-1]  ‘ Pytho’  >>> a[-7:0]  >>> a[-7:2]  ‘’ |

**1-2. 문자열을 바꾸고 싶을 때**

str 은 immutable 한 자료형이기 때문에 직접 바꿀 수 없다. cf) replace 함수도 사용가능.

즉, str은 CRUD 불가능.

|  |
| --- |
| >>> a = "Pithon"  >>> a[1] = "y"  ERROR  >>> a[:1] + "y" + [2:]  'Python' |

**1-3. 문자열 포매팅**

포맷 코드: %s %c %d %f %o %x %%

|  |
| --- |
| >>> number = 10  >>> day = "three"  >>> "I ate %d apples. so I was sick for %s days." % (number, day)  'I ate 10 apples. so I was sick for three days.'  >>> "I ate %d apples. so I was sick for %s days." % (day, number)  TYPE ERROR |

순서와 타입에 주의해야 한다.

공백 부여하기도 가능하다.

|  |
| --- |
| >>> "%10sjane" % "hi"  ' hijane'  >>> "%-10sjane" % "hi"  'hi jane' |

소수점을 절삭하고 보여주기도 가능하다.

|  |
| --- |
| >>> "%0.4f" % 3.42134234  '3.4213'  >>> "%10.4f" % 3.42134234  ' 3.4213' |

**1-3-2. format 함수로 포매팅**

넣는 순서를 인덱스로 지정 가능하다.

|  |
| --- |
| >>> "I ate {0} apples. so I was sick for {1} days.".format(number, day)  'I ate 9 apples. so I was sick for five days.'  >>> "I ate {1} apples. so I was sick for {0} days.".format(number, day)  'I ate five apples. so I was sick for 9 days.' |

이름으로 넣기도 가능하다.

|  |
| --- |
| >>> "I ate {number} apples. so I was sick for {day} days.".format(number=10, day=3)  'I ate 10 apples. so I was sick for 3 days.' |

인덱스와 이름을 혼용 가능.

|  |
| --- |
| >>> "I ate {1} apples. so I was sick for {day} days.".format(10, 4, day=3)  'I ate 4 apples. so I was sick for 3 days.' |

왼쪽 정렬, 오른쪽 정렬, 가운데 정렬, 공백 채우기, 소수점 표현

|  |
| --- |
| >>> "{0:<10}".format("hi")  'hi ' |
| >>> "{0:>10}".format("hi")  ' hi' |
| >>> "{0:^10}".format("hi")  ' hi ' |
| >>> "{0:=^10}".format("hi")  '====hi====' |
| >>> "{0:0.4f}".format(3.14159265)  '3.1416' |

**1-3-3. f 문자열 포매팅 ★★★★★**

파이썬 3.6부터 가능.  
변수를 계산식과 함께 사용 가능. 인덱싱도 가능하며 dict의 key value도 참조 가능.

|  |
| --- |
| >>> name = "홍길동"; age = 30  >>> f"나의 이름은 {name}입니다. 나이는 {age}입니다."  '나의 이름은 홍길동입니다. 나이는 30입니다.' |
| >>> f'나는 내년이면 {age + 1}살이 된다.'  '나는 내년이면 31살이 된다.' |
| >>> d = {"name" : "홍길동", "age" : 30} **★★★**  >>> f"나의 이름은 {d["name"]}입니다. 나이는 {d["age"]}입니다."  '나의 이름은 홍길동입니다. 나이는 30입니다.' |
| >>> l = ["a", 123, "홍길동", [2,3], 30]  >>> f"나의 이름은 {l[2]}입니다. 나이는 {l[4]}입니다."  '나의 이름은 홍길동입니다. 나이는 30입니다.' |

정렬, 공백채우기, 소수점 표현

|  |
| --- |
| >>> f'{"hi":<10}' # 왼쪽 정렬  'hi '  >>> f'{"hi":>10}' # 오른쪽 정렬  ' hi'  >>> f'{"hi":^10}' # 가운데 정렬  ' hi ' |
| >>> f'{"hi":=^10}'  '====hi====' |
| >>> y = 3.14159265  >>> f"{y:0.4f}"  '3.1416'  >>> f"{y:10.4f}" # 소수점 4자리까지 표현하고 총 자리수를 10으로 맞춤  ' 3.1416' |

**1-4. 문자열 기초 함수들 ★★★★★**

count find index join upper lower replace split strip lstrip rstrip

**1-5. 이스케이프 코드**

\n \t 는 자주 쓰이지만  
\r \f \a \b \000은 잘 안 쓰인다.

**2. list 리스트**

a = list() a = [] 비어있는 리스트 생성.

인덱싱과 슬라이싱 가능.

+ 와 \* 연산 가능.

수정과 삭제가 가능하다. CRUD 가능.

|  |
| --- |
| >>> a = [1, 2, 3]  >>> a[2] = 9  >>> a  [1, 2, 9] |
| >>> del a[1]  >>> a  [1, 9] |
| >>> a = [1, 2, 3, 4, 5]  >>> del a[2:] # 여러 요소를 한 번에 삭제하기 가능  >>> a  [1, 2] |

**2-2. 리스트 기초 함수들**

append sort reverse index insert remove pop count extend

**3. tuple 튜플**

리스트는 요소의 생성, 삭제, 수정이 자유롭지만, 튜플은 요소를 **삭제, 수정할 수 없다**.

a = tuple() a = () 비어있는 튜플 생성.

a = (3,) 요소가 한 개인 튜플을 만들 때.

인덱싱과 슬라이싱 가능.

+ 와 \* 연산 가능.

**튜플은 요소를 변경할 수 없기 때문에 append, sort, insert, remove, pop, extend와 같은 내장 함수가 없다.**

**4. dict 딕셔너리**

요즘 사용하는 대부분의 언어도 이러한 대응 관계를 나타내는 자료형을 가지고 있는데 이를 딕셔너리라고 하고, ‘연관 배열(associative array)’또는 ‘해시(hash)’라고도 한다.

Key 와 Value를 한 쌍으로 가지는 자료형으로 사전형, 개인정보형으로 적합하다.

**4-2. 딕셔너리 쌍 추가 및 삭제하기**

|  |
| --- |
| >>> a = {1: 'a'}  >>> a[2] = 'b'  >>> a  {1: 'a', 2: 'b'} |
| >>> del a[1]  >>> a  {2: 'b'} |

Key는 중복을 허용하지 않는다.

때문에 Key로 mutable인 List는 사용할 수 없지만, immutable인 Tuple은 사용할 수 있다.

|  |
| --- |
| >>> a = {[1,2] : 'hi'}  ERROR |
| >>> a = {(1,2) : 'hi'}  >>> a  {(1, 2): 'hi'} |

**4-3. 딕셔너리 기초 함수들**

keys values items clear get in

keys , values , items 는 각각 dict\_keys 객체 , dict\_values , dict\_items 객체를 리턴한다.

때문에 이들을 리스트로 받고 싶다면 list( myDict.keys() ) 이렇게 list로 감싸야 한다.

단, 리스트로 변환하지 않아도 **반복문에 직접 넣어서 하나씩 참조할 수 있다.** ★

items 함수는 Key와 Value를 한 쌍의 **튜플로 묶어 리턴**한다. ★

|  |
| --- |
| >>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}  >>> a.items()  dict\_items([('name', 'pey'), ('phone', '010-9999-1234'), ('birth', '1118')]) |

Key에 해당하는 Value를 가져올 때 get을 사용하는게 더 유리하다.

**get은 존재하지 않는 Key를 가져올 때 None을 리턴한다. 또한 디폴트값을 리턴하게 할 수 있다.** ★

|  |
| --- |
| >>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}  >>> a["notExist"]  ERROR  >>> a.get("notExist") == None  True |
| >>> a.get("notExist", "default\_value")  'default\_value' |

in 은 해당 키가 딕셔너리 안에 있는 지를 bool로 리턴한다.

|  |
| --- |
| >>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}  >>> "name" in a  True |

**5. set 집합**

중복 불가. 순서가 없다.(unordered)

s = set() 비어있는 set 만들기.

set을 만드는 방법

|  |
| --- |
| >>> s1 = set([1, 2, 3])  >>> s1  {1, 2, 3} |
| >>> s2 = set("Hello")  >>> s2  {'e', 'o', 'H', 'l'} |

**5-2. set 은 교집합, 합집합, 차집합 구할 때 용이하다. ★★★**

|  |
| --- |
| >>> s1 = set([1, 2, 3, 4, 5, 6])  >>> s2 = set([4, 5, 6, 7, 8, 9])  >>> s1 & s2 # 교집합  {4, 5, 6}  >>> s1.intersection(s2) # 교집합  {4, 5, 6} |
| >>> s1 | s2 # 합집합  {1, 2, 3, 4, 5, 6, 7, 8, 9}  >>> s1.union(s2) # 합집합  {1, 2, 3, 4, 5, 6, 7, 8, 9} |
| >>> s1 – s2 # 차집합  {1, 2, 3}  >>> s1.difference(s2) # 차집합  {1, 2, 3} |
| >>> s1 ^ s2 # 합집합에서 교집합을 뺀. 배타 집합  >>> s1.symmetric\_difference(s2) # 합집합에서 교집합을 뺀. 배타 집합  {1, 2, 3, 7, 8, 9} |

**5-3. set 기본 함수들**

add update remove

**5-4. set 에 인덱스 쓰기**

순서가 없기 때문에 인덱스로 참조 불가능 하지만, list로 바꾸는 꼼수를 쓰면 가능하다.

print( list(s1)[인덱스] ) 이 때, 오름차순으로 정렬돼있다.

**6. 자료형의 값을 저장하는 공간, 변수**

변수란 객체를 가리키는 것.

|  |
| --- |
| >>> a = [1, 2, 3]  >>> id(a) # 메모리의 주소를 반환한다.  2074172949376 |
| >>> b = a  >>> id(b)  2074172949376  >>> a is b # a와 b가 같은 대상을 가리킨다는 것을 확인할 수 있다.  True  >>> a[0] = 4  >>> b  [4, 2, 3] |
| >>> from copy import copy  >>> b = copy(a) # 값을 복사해서 다른 메모리에 저장.  >>> a[0] = 9  >>> b  [4, 2, 3] |

변수를 만드는 방법들과 서로 바꾸기

|  |
| --- |
| >>> a, b = ('python', 'life')  >>> (a, b) = 'python', 'life'  >>> [a, b] = ['python', 'life'] |
| >>> a = 3 ; b = 5  >>> a, b = b, a  >>> print(a, b)  5 3 |

|  |
| --- |
|  |
| **2. 제어문** |

**1. if**

요즘 파이썬 커뮤니티에서는 들여쓰기를 할 때 공백 문자 4개를 사용하는 것을 권장한다. 또한 파이썬 에디터는 대부분 탭 문자로 들여쓰기를 하더라도 탭 문자를 공백 문자 4개로 자동 변환하는 기능을 갖추고 있다.

**1-1. 비교 연산자와 연산자**

< > == != >= <=

and or not ex) not x x가 거짓이면 참이다.

in not in 리스트, 튜플, 문자열

|  |
| --- |
| >>> a = [1,2,3]  >>> 1 in a  True  >>> 4 not in a  True  >>> "y" not in "Python"  False |

**1-2. 조건부 표현식 ★★★★★**

아래 두 표현은 서로 같다.

|  |
| --- |
| if score >= 60:  message = "success"  else:  message = "failure" |
| message = "success" if score >= 60 else "failure" |

변수 = 조건문이\_참인\_경우의\_값 if 조건문 else 조건문이\_거짓인\_경우의\_값

**2. while**

조건이 참인 동안에 계속 반복되므로 탈출할 수 있는 조건이 필요하다.

|  |
| --- |
| treehit = 0  while treehit<10:  treehit += 1  print(f"나무를 {treehit}번 째 찍는 중")  if treehit == 10:  print("나무가 쓰러짐") |
| a = 0  while a < 10:  a += 1  if a % 2 == 0: continue  print(a) |

**3. for**

for 변수 in 리스트,튜플,문자열:

응용

|  |
| --- |
| >>> a = [(1,2), (3,4), (5,6)]  >>> for (first, last) in a:  ... print(f"{first} + {last} = {first+last}")  1 + 2 = 3  3 + 4 = 7  5 + 6 = 11 |

**3-2. 리스트 컴프리헨션 ★★★★★**

아래 두 표현은 같다.

|  |
| --- |
| >>> a = [1,2,3,4]; result = []  >>> for num in a:  ... result.append(num\*3)  >>> result  [3, 6, 9, 12] |
| >>> result = [num\*3 for num in a]  >>> result  [3, 6, 9, 12] |

응용

|  |
| --- |
| >>> result = [num\*3 for num in a if num%2==0] # 간단한 조건문도 넣을 수 있다.  >>> result  [6, 12] |

**[표현식 for 항목 in 반복\_가능\_객체 if 조건문] ★★★★★**

이를 2개 이상 사용하는것도 가능하다.

|  |
| --- |
| [표현식 for 항목1 in 반복\_가능\_객체1 if 조건문1  for 항목2 in 반복\_가능\_객체2 if 조건문2  ...  for 항목n in 반복\_가능\_객체n if 조건문n] |

continue와 pass 차이

|  |
| --- |
|  |
| **3. 함수와 입출력** |

**1. 함수**

반복되는 부분이 있을 경우, 함수를 호출하면 편하다.

프로그램을 기능 단위의 함수로 분리해 놓으면 흐름을 보기 편하다.

**1-1. parameter와 argument 매개변수와 인수**

argument = 함수를 호출할 때 입력하는 값.

parameter = 함수 호출로 들어온 값을 받는 변수.

**1-2. parameter를 직접 지정하기 및 parameter에 초깃값 주기 ★★★**

|  |
| --- |
| ＞＞＞ def func(a, b, c):  ㆍㆍㆍ return a-b+c  >>> func(10, c=1, b=2) # **parameter를 직접 지정한 argument는 무조건 끝으로 가야 한다.**  9 |
| ＞＞＞ def func(a, c, b=2): # **parameter를 직접 지정한 argument는 무조건 끝으로 가야 한다.**  ㆍㆍㆍ return a-b+c  >>> func(10, 3)  11 |

**1-3. 여러 개의 입력값을 받기 ★★★★★**

**\* 을 붙이면, 입력값을 모두 모아 튜플로 만든다.**

|  |
| --- |
| def add\_mul(choice, \*inputs):  res = 0  if choice == "add":  for i in inputs:  res += i  elif choice == "mul":  res = 1  for i in inputs:  res \*= i  else: return "ERROR"  return res |
| >>> add\_mul("add", 1, 2, 3, 4, 5)  15  >>> add\_mul("mul", 1, 2, 3, 4, 5)  120 |

**1-4. 키워드 매개변수 kwargs ★★★**

매개변수 앞에 \*\* 을 붙이면 매개변수는 딕셔너리가 되고, 모든 Key = Value 형태의 입력값이 이 딕셔너리에 저장된다.

|  |
| --- |
| def print\_kwargs(\*\*kwargs):  print(kwargs) |
| >>> print\_kwargs(a = 1)  {'a': 1}  >>> print\_kwargs(name = "kim", age = 13)  {'name': 'kim', 'age': 13} |

**1-5. 함수의 리턴값은 언제나 한 개다. 리턴을 함수 탈출용으로 사용할 수 있다.**

여러 값을 같이 리턴하면, 튜플로 리턴된다.

|  |
| --- |
| def func(a,b):  return a + b, a \* b |
| >>> func(7, 12)  (19, 84)  >>> type(func(7, 12))  <class 'tuple'> |

함수를 끝마치고 나가고 싶을 땐 return을 단독으로 쓰면 된다.

|  |
| --- |
| def say\_nick(nick):  if nick == "바보":  return  print("나의 별명은 %s 입니다." % nick) |

**1-6. 함수 안에서 함수 밖의 변수에 접근하기 ★★★★★**

1-6-1. return 사용하기

|  |
| --- |
| a = 1 def vartest(a):  a = a +1  return a  a = vartest(a) print(a) |

1-6-2. global 사용하기

|  |
| --- |
| a = 1 def vartest():  global a  a = a+1  vartest() print(a) |

**2. lambda 예약어**

lambda는 함수를 생성할 때 사용하는 예약어로, def와 역할이 같다.

**def를 써야할 정도로 복잡하지 않은 함수를 한 줄로 간결하게 만들 때 사용. ★★★**

**혹은 def를 사용할 수 없는 곳에 사용.**

함수\_이름 = lambda 매개변수1, 매개변수2, … : 리턴될\_값을\_매개변수를\_활용해서\_표현하기

|  |
| --- |
| >>> add = lambda a, b : a+b  >>> result = add(3, 4)  >>> print(result)  7 |

**3. 입출력**

input() input은 **입력되는 모든 것을 str 로 받는다.**

print()

**3-2. 파일을 생성하고 읽고 쓰기**

파일\_객체 = open(파일\_이름, 파일\_열기\_모드)

열기 모드의 종류: r - 읽기 모드 , w - 쓰기 모드 , a - 추가 모드

파일 생성

|  |
| --- |
| f = open("새파일.txt", 'w') f.close() |
| f = open("C:/doit/새파일.txt", 'w') # **파이썬에서 경로를 표시할 땐 \ 대신 / 을 사용한다.**  f.close() |

쓰기 모드로 열어서 내용 쓰기

|  |
| --- |
| f = open("새파일.txt", 'w') for i in range(1, 11):  f.write(f"{i}번째 줄입니다.\n") f.close() |

readline 으로 읽기

|  |
| --- |
| f = open("새파일.txt", 'r') while True:  line = f.readline() # 한 줄씩 읽다가 더 이상 읽을게 None이면 반복문을 탈출한다.  if not line: break  print(line) f.close() |

readlines 로 읽기

|  |
| --- |
| f = open("새파일.txt", 'r') lines = f.readlines() # 각 줄을 요소로 만들어 리스트에 넣는다 for line in lines:  print(line) f.close() |

read 로 읽기

|  |
| --- |
| f = open("새파일.txt", 'r') data = f.read() #read 는 파일 내용 전체를 문자열로 리턴한다. 고로 data는 파일 전체 내용. print(data) f.close() |

파일 객체를 for 문에 받아 읽기

|  |
| --- |
| f = open("새파일.txt", 'r') for line in f: #파일 객체 자체를 활용해 줄 단위로 읽을 수 있다.  print(line) f.close() |

파일에 내용 추가하기

|  |
| --- |
| f = open("새파일.txt", 'a') for i in range(11, 21):  f.write(f"{i}번째 줄입니다.\n") f.close() |

with와 함께 쓰면 f.close()로 번거롭게 닫을 필요가 없다.

|  |
| --- |
| with open("새파일.txt", "w") as f:  f.write("Life is too short, you need python") |

**3-3. 프로그램의 입출력**

?????

|  |
| --- |
|  |
| **4. 추가 예제들** |

**1. 자료형**

h\_format.py

|  |
| --- |
| >>> print("%d %3f %3.3f %15f"%(3.14,3.14,3.14,3.14))  3 3.140000 3.140 3.140000 |
| %d가 내림함.  %3f로 3칸만 준다고 했으나 .이 없으면 소수점 6자리까지 출력해야해서 3칸 이상을 출력했다.  %3.3f 소수점 3자리까지만 출력했지만 3칸을 초과해서 보여준다.  %15f 소수점을 다 출력해도 여유가 있으니 우측정렬 후 공간이 빈다. |

**제로 패딩** : 0으로 빈 칸 채우는 것. 시계열데이터나 이미지 같은 자료에서 0으로 빈 칸을 채우는 것은 중요하다.

|  |
| --- |
| # 2진화 → 10진화 bin(value) → **int(bin(value), base = 0) base 값을 안 넣어주면 오류남.** num =42 **# 0x 0o 0b 로 시작하면 base에 0을 넣어줘야만 함.** f= "The number is {:010d}".format(num) print(f, type(f)) f= "The number is {:0>10d}".format(num) print(f, type(f)) f= "Binary is {:b}".format(num) print(f, type(f)) f = "Binary is {:08b}".format(num) print(f, type(f)) t = bin(num) print(t, type(t)) ### bin()을 쓰면 0b가 포함된 str로 출력. |
| The number is 0000000042 <class 'str'>  The number is 0000000042 <class 'str'>  Binary is 101010 <class 'str'>  Binary is 00101010 <class 'str'>  0b101010 <class 'str'> |

\*\*로 dict에서 바로 참조

|  |
| --- |
| person={'name':'홍길동' ,'age':20} f="person : Name:{name} , age:{age}".format(\*\*person) |
| person : Name:홍길동 , age:20 |

\*로 tuple에서 바로 참조

|  |
| --- |
| point=(1,2,3,4) f= "Point: ( {},{},{},{} )".format(\*point) print(f) point01 = (11, 2,3,4) f = "Point: ( {},{},{},{}, {},{},{},{} )".format(\*point, \*point01) print(f) a, b, \*c = 1,2,3,4,5,6,7,7,8 print(a,b,c, type(c)) ### 리스트로 출력된다!!! 주의 |
| Point: ( 1,2,3,4 )  Point: ( 1,2,3,4, 11,2,3,4 )  1 2 [3, 4, 5, 6, 7, 7, 8] <class 'list'> |

날짜 서식

|  |
| --- |
| import datetime # 날짜 시간 모듈  #날짜서식 date(year, month, day) --> date object date = datetime.date(2023,7,7) f= "Date : {: %Y-%m-%d}".format(date) print(f) print("today = ", datetime.datetime.today()) print("Today : {: %Y-%m-%d}".format(datetime.datetime.today())) #### 아주 좋다 ''' datetime 모듈의 datetime 클래스의 today 함수와 now()함수는 timezone에 따라 달라진다. 1. datetime.datetime.today() → 현재 지역대 기반으로 datetime객체를 리턴한다. (나라(로컬) 시간대를 따름) 2. datetime.datetime.now() → 현재 PC 기반으로 datetime객체를 리턴한다. (현 컴퓨터 시간대를 따름) ''' |
| Date : 2023-07-07  today = 2023-11-14 01:13:09.116979  Today = 2023-11-14 |

Doit2-6.py

|  |
| --- |
| list1 = [0, 1, 2, 3, 4, 5] print( 3 in list1) # True ----------> in (data or data or,,,,,) print( 7 not in list1) # True |
| True  True |

비트 연산자 Doit2-4.py

|  |
| --- |
| a=10; # 1010 print(a) b=2; # 10 print(b)  print(a & b ) # 0010  print(a | b ) # 1010  print(a ^ b ) # 1000 print(a <<1 ) # 10100  print(a >>1 ) # 101  print(~ a ) # -11 |