Python_01

Programming(== Coding)

1. Programming?

- 소프트웨어를 개발하기 위한 과정
- 컴퓨터에게 명령하는 적절한 수행 절차를 정의하고 이를 프로그래밍 언어로 표현하는 과정
- 컴퓨터 내에 그 순서가 정해진 명령어 순서를 진행하는 것
- 계산, 저장, 반복이 뛰어남

2. 프로그래밍 과정

1. 컴퓨터에게 시키고 싶은 일을 정한다.
2. 컴퓨터가 이해할 수 있도록 수행 절차를 정의해서 표현한다. (알잘딱깔센 X)
3. 적절한 프로그래밍 언어를 선택하고, 언어를 이용해서 절차를 기술한다.
4. 발생하는 오류를 수정한다.
    a. 구문 오류(syntax error) == 문법 error, 실행 error
    b. 논리 오류(semantic error) == 문법 pass, 실행 pass, 결과 error

3. Computational Thinking

컴퓨팅 사고 또는 전산적 사고는 컴퓨터가 효과적으로 수행할 수 있도록 문제를 정의하고 그에 대한 답을 기술하는 것이 포함된 사고 과정 일체를 일컫는다.

1. 컴퓨터의 특성을 잘 이해한다 (understanding computer)
2. 문제 해결 능력을 기른다. (problem solving)
   (논리적 사고 == 작은 문제로 쪼개기)
3. 프로그래밍 언어에 능숙해진다. (trial & error)

Python(3.9.12)

1. 프로그래밍 언어란?

- 기계어(0과 1로 모든 것을 표현 == 2진법)의 대안으로 사람이 이해할 수 있는 새로운 언어 개발
- 사람이 이해할 수 있는 문자로 구성
- 기본적인 규칙과 문법이 존재

2. 프로그래밍 언어의 구성

소스 코드 : 프로그래밍 언어로 작성된 프로그램

번역기(interpreter/compiler)

- 소스 코드를 컴퓨터가 이해할 수 있는 기계어로 번역
  - 인터프리터 : 기계어로 한줄씩 바꿔주고 실행
  - 컴파일러 : 모두 기계어로 변환하여 실행 ex) .exe
- 파이썬의 경우 인터프리터를 사용

3. Python의 특징

- 다른 프로그래밍 언어에 비해 문법이 간단하며, 엄격하지 않음
- 별도의 데이터 타입 지정이 필요 없으며, 재할당이 가능함
- 문장을 구분할 때 중괄호를 사용하지 않고 들여쓰기를 사용함
- 소스코드를 기계어로 변환하는 컴파일 과정 없이 바로 실행이 가능함
- 객체 지향 프로그래밍 언어로 모든 것이 객체로 구현되어 있음

>> git bash

```
SSAFY@DESKTOP-DOGVPUB MINGW64 /c/SSAFY/python
$ python -V
Python 3.9.13

SSAFY@DESKTOP-DOGVPUB MINGW64 /c/SSAFY/python
$ python
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello")
hello
>>> my_age = 25
>>> print("제 나이는", my_age, "입니다")
제 나이는 25 입니다
```

> CLI : 명령어를 통해 사용자와 컴퓨터가 상호 작용하는 방식

4. Python 개발 환경 종류

- IDE (Intergrated Development Environment)

  - 통합 개발 환경의 약자로 개발에 필요한 다양하고 강력한 기능들을 모아둔 프로그램
  - 보통 개발은 IDE로 진행

- Jupyter Notebook

  - 문법 학습을 위한 최적을 도구로, 소스 코드와 함께 실행 결과와 마크다운 저장 가능
  - open source 기반의 웹 플랫폼 및 어플리케이션으로, 파이썬을 비롯한 다양한 프로그래밍 언어를 지원하며 셀 단위의 실행이 가능한 것이 특징

- IDLE (Intergrated Development and Learning Environment)

Python 기초문법

1. Input & Output

```
# 입력을 할 때 함수
s = input() # >> 1
print(type(s)) # <class 'str'>
print(int(s) + 1) # 2

n = int(input()) # >> 1
print(type(n)) # <class 'int'>

a = input().split(" ") # >> a b c
print(a) # ['a', 'b', 'c']

m = list(map(str, ['a', 'b', 'c']))
print(m) # ['a', 'b', 'c']

i = list(map(int, ['10', '20', '30']))
print(i) # [10, 20, 30]
```

2. 변수와 식별자

variable(변수)

- 데이터를 저장하기 위해서 사용
- 변수를 사용하면 복잡한 값들을 쉽게 사용할 수 있음(추상화)
- 동일 변수에 다를 데이터를 언제든 할당(저장)할 수 있기 때문에, '변수'라고 불림

변수를 사용해야 하는 이유

- 코드의 가독성 증가
- 숫자를 직접 적지 않고, 의미 단위로 작성 가능
- 코드 수정이 용이해짐 - ex) 아메리카노 가격이 변경되더라도 1곳만 수정하면 됨

변수의 할당

- 변수는 할당 연산자 (=)를 통해 값을 할당(assignment)
- 같은 값을 동시에 할당할 수 있음
- 다른 값을 동시에 할당할 수 있음

실습 문제

문제)

x = 10, y= 20 일 때, 각각 값을 바꿔서 저장하는 코드
를 작성해 보시오

방법1)

임시 변수 활용

방법2)

Pythonic!

```python
x = 10
y = 20

# method 1
tmp = x
x = y
y = tmp

print(x, y)

# method 2
y, x = x, y

print(x, y)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\SSAFY\python\python01> python .\python.py
20 10
10 20
```

식별자(Identifiers)

변수의 이름을 식별자라고 함(변수, 함수, 클래스 …)

변수 이름 규칙

- 식별자의 이름은 영문 알파벳, 언더스코어(_), 숫자로 구성
- 첫 글자에 숫자가 올 수 없음
- 길이 제한이 없고, 대소문자를 구별
- 다음 키워드(keywords)는 예약어(reserved words)로 사용할 수 없음

```
import keyword
print(keyword.kwlist)

#출력 결과
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async',
'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

- 내장 함수나 모듈 등의 이름도 사용하지 않아야 함
  - 동작을 예상 할 수 없게 임의로 값을 할당하게 되므로 범용적이지 않은 코드가 됨

3. 주석(comment)

```
#
//
'''
'''
```

- 코드에 대한 쉬운 이해
- 유지보수 용이
- 협업 용이

4. 연산자

산술 연산자(Arithmetic Operator)

- 기본적인 사칙연산 및 수식 계산

연산자	내용
+	덧셈
-	뺄셈
*	곱셈
/	나눗셈

- 기본적으로 수학에서 우선순위와 같음
- 괄호가 가장 먼저 계산되고, 그 다음에
  곱하기(*)와 나누기(/)가
  더하기(+)와 빼기(-)보다 먼저 계산됨

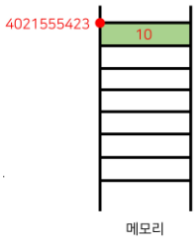
//	몫
**	거듭제곱

5. 자료형

메모리

HDD, SSD <<<<<<< Ram

- 데이터 10을 컴퓨터가 기억하는 과정
  1. 10을 저장할 공간을 메모리에 만들고
  2. 저장할 공간에 대한 주소를 할당받는다
  3. 할당 받은 주소를 기억했다가(4021555423)
  4. 10이라는 데이터를 해당 주소로 찾아가서 저장한다.
  5. 이후에 10이 필요해지면 해당 주소로 가서 읽어온다.



4021555423 ⇒ 주소값을 기억하기가 어려움

변수 선언(variable declaration) == '변수'를 사용하여 기억하기 쉬운 이름으로 저장하자!

자료형

	자료형	저장 모델	변경 가능성	접근 방법
수치형	int, float, complex	Literal	Immutable	Direct
문자열	str	Container		Sequence
튜플	tuple			
리스트	list		Mutable	
사전	dict			Mapping
집합	set			Set

```
a = 'abc'
print(id(a)) # 1721886571248
a = a + 'd'
print(id(a)) # 1721886602928
b = 'abc'
print(id(b)) # 1721886571248
```

> - 프로그래밍에서 변수는 메모리의 주소를 기억하는 이름이다.
> - 우리는 변수를 이용해서 데이터를 기억한다.
> - 자료형마다 차지하는 메모리의 크기가 다르다.

```
PS C:\SSAFY\python\python01> python .\my_age.py
제 나이는 26 입니다
2999751437392
```

```python
my_age = 26
print("제 나이는", my_age, "입니다")

memory_addr = id(my_age)
print(memory_addr)
```

6. 수치형(Numeric Type)

정수 자료형(int)

- 0, 100, -200과 같은 정수를 표현하는 자료형
- 일반적인 수학 연산(사칙 연산) 가능

실수 자료형(float)

- 0.1, 100.0, -0.001과 같은 유리수와 무리수를 포함하는 '실수'를 다루는 자료형
- **실수 연산 시 주의할 점 (부동 소수점)**
  - 실수의 값을 처리할 때 의도하지 않은 값이 나올 수 있음

```
print(3.2 - 3.1) # 0.10000000000000009
```

  - 원인은 부동 소수점 때문
    - 컴퓨터는 2진수 사용, 사람은 10진법 사용
    - 이때 10진수 0.1은 2진수로 표현하면 0.0001100110011001100110… 같이 무한대로 반복
    - 무한대 숫자를 그대로 저장할 수 없어서 사람이 사용하는 10진법의 근사값만 표시
    - 이런 과정에서 예상치 못한 결과가 나타남 == Floating point rounding error

해결방법 : 값 비교하는 과정에서 정수가 아닌 실수면 주의할 것!
(매우 작은 수보다 작은지를 확인하거나 math 모듈 활용)

```
a = 3.2 - 3.1
b = 1.2 - 1.1

# 1. 임의의 작은 수 활용
print(abs(a-b) <= 1e-10) # True

# 2. math 모듈 사용
import math
print(math.isclose(a,b)) # True
```

진수 표현

- 여러 진수로 표현 가능
  - 2진수(binary) : 0b

```
print(0b10)
```

```
2
```

  - 8진수(octal) : 0o

```
print(0o30)
```

```
24
```

  - 16진수(hexadecimal) : 0X

```
print(0x10)
```

```
16
```

7. 문자열 자료형(String Type)

- 모든 문자는 str 타입
- 문자열은 작은따옴표나 큰따옴표를 활용하여 표기
  - 문자열을 묶을 때 동일한 문장 부호 사용
  - PEP8에서는 소스코드 내에서 하나의 문장 부호를 선택하여 유지하도록 함
- 중첩따옴표
  - 작은따옴표가 들어 있는 경우는 큰따옴표로 문자열 생성
  - 큰따옴표가 들어 있는 경우는 작은따옴표로 문자열 생성
- 삼중 따옴표
  - 따옴표 안에 따옴표를 넣을 때 여러 줄을 나눠 입력할 때 편리
- Escape sequence
  - 역슬래시 뒤에 특정 문자가 와서 특수한 기능을 하는 문자 조합(제어 시퀀스)

예약 문자	내용
\n	줄 바꿈
\t	탭
\r	캐리지 리턴(커서를 여기서 다시 시작) print('aaaaa\rbbb') # bbbaa
\o	널(NULL)
\\	\
\'	단일인용부호(')
\"	이중인용부호(")

- 문자열 연산
  - 덧셈

```
print('Hello' + 'world')
# Helloworld
```

  - 곱셈

```
print('Python' * 3)
# PythonPythonPython
```

- String Interpolation
  - f-strings: python 3.6+

```
name = 'Seohyun'
score = 4.5

print(f'Hello, {name}! 성적은 {score}')
# Hello, Seohyun! 성적은 4.5
```

8. None

- 파이썬 자료형 중 하나

- 값이 없음을 표현하기 위해 None 타입 존재
- 일반적으로 반환 값이 없은 함수에서 사용하기도함

9. 불린형(Boolean)

- 논리 자료형으로 참과 거짓을 표현하는 자료형
- True 또는 False를 값으로 가짐
- 비교 / 논리 연산에서 활용됨

비교 연산자

- 수학에서 등호와 부등호와 동일한 개념
- 주로 조건문에 사용되며 값을 비교할 때 사용
- 결과는 True / False 값을 반환함

```
print(3 > 6)
print(3.0 == 3)
print(3 >= 0)
print('3' != 3)
print('Hi' == 'hi')
```

```
False
True
True
True
False
```

연산자	내용
<	미만
<=	이하
>	초과
>=	이상
==	같음
!=	같지않음
is	객체 아이덴티티(OOP)
is not	객체 아이덴티티가 아닌 경우

논리 연산자

- 여러가지 조건이 있을 때
  - 모든 조건을 만족하거나(And),
    여러조건 중 하나만 만족해도 될 때(or)
    특정 코드를 실행하고 싶을 때 사용
  - 일반적으로 비교연산자와 함께 사용됨

연산자	내용
A and B	A와 B 모두 True시 True
A or B	A와 B 모두 False시 False
Not	True를 False로 False를 True로

논리 연산자 주의할 점/ not 연산자

- Falsy : False는 아니지만 False로 취급 되는 다양한 값
  - 0, 0.0, (), {}, [], None, “”(빈문자열)
- 논리 연산자도 우선순위가 존재
  - not, and, or 순으로 우선순위가 높음

논리 연산자의 단축 평가

```
grade = [60, 70, 80]
print(grade[True]) # 70
print(grade[False]) # 60
```

- 결과가 확실한 경우 두번째 값은 확인하지 않고 첫번째 값 반환
- and 연산에서 첫번째 값이 False인 경우 무조건 False ⇒ 첫번째 값 반환
- or 연산에서 첫번째 값이 True인 경우 무조건 True ⇒ 첫번째 값 반환

- 0은 False, 1은 True

컨테이너

- 여러개의 값(데이터)을 담을 수 있는 것(객체)으로, 서로 다른 자료형을 저장할 수 있음. ex) List
- 컨테이너의 분류
  - 순서가 있는 데이터(Ordered) vs 순서가 없는 데이터(Unordered)
  - 순서가 있다 != 정렬되어 있다.

		리스트	grade = [50, 60, 70] grade[1] = 50 # [50, 50, 70]
	시퀀스형	튜플	
컨테이너		레인지	
	비시퀀스형	세트	grade = [50, 60, 70] grade[1] = 50 print(set(grade)) # {50, 70}
		딕셔너리	

리스트(List)

- 리스트는 여러개의 값을 순서가 있는 구조로 저장하고 싶을 때 사용

```
dust = [58, 40, 70, 60, 120, 54, 23, 50]
dust[2] = 30
print(dust)
# [58, 40, 30, 60, 120, 54, 23, 50]
```

- 리스트는 대괄호([]) 혹은 list()를 통해 생성
  - 파이썬에서는 어떠한 자료형도 저장할 수 있으며, 리스트 안에 리스트도 넣을 수 있음
  - 생성된 이후 내용 변경이 가능 → 가변 자료형
  - 이러한 유연성 때문에 파이썬에서 가장 흔히 사용
- 순서가 있는 시퀀스로 인덱스를 통해 접근 가능
  - 값에 대한 접근 리스트 list[i]

```
dust = [58, 40, 70, 60, 120, 54, 23, 50]
temp = [30, 40, 50]
dust[1] = temp
print(dust)
# [58, [30, 40, 50], 70, 60, 120, 54, 23, 50]
print(dust[1][1])
# 40
```

실습

```
boxes = ['A', 'B', ['apple', 'banana', 'cherry']]

print(len(boxes))
  # 3
print(boxes[2])
  # ['apple', 'banana', 'cherry']
print(boxes[2][-1])
  # cherry
print(boxes[-1][1][0])
  # b
```

튜플 (Tuple)

1. 튜플 정의

- 튜플은 여러개의 값을 순서가 있는 구조로 저장하고 싶을 때 사용
  - 리스트와의 차이점은 생성 후, 담고 있는 값 변경이 불가(불변 자료형)
- 항상 소괄호 형태로 사용

2. 튜플의 생성과 접근

- 소괄호(()) 혹은 tuple()을 통해 생성
- 튜플은 수정 불가능한(immutable) 시퀀스로 인덱스로 접근 가능
- 값에 대한 접근은 my_tuple[i]

3. 튜플 생성 주의사항

- 단일 항목인 경우
  - 하나의 항목으로 구성된 튜플은 생성 시 값 뒤에 쉼표를 붙여야 함
- 복수 항목의 경우
  - 마지막 항목에 붙은 쉽표는 없어도 되지만, 넣는 것을 권장(Trailing comma)

레인지(Range)

- 기본형 : range(n)
  - 0부터 n-1까지의 숫자의 시퀀스
- 범위 지정: range(n, m)
  - n부터 m-1까지의 숫자의 시퀀스

```
print(range(10)) # range(0, 10)
print(list(range(10))) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
a = list(range(10))
print(a[6]) # 6
print(type(range(10))) # <class 'range'>
```

슬라이싱 연산자

시퀀스를 특정 단위로 슬라이싱

> [시작 : 끝 : 스텝]
> [: 끝]
> [:]
> [: : 2]
> [-2:]
> [: : -1]
> a == a [::-1] **팰린드롬**(palindrome)

```
print('12345678'[2:6:2]) #35
print('12345678'[:5]) #12345
print('12345678'[:]) #12345678
print('12345678'[::2]) #1357
print('12345678'[-2:]) #78
print('12345678'[::-1]) #87654321
```

💡 **팰린드롬 (palindrome) or 회문**
거꾸로 읽어도 제대로 읽는 것과 같은 문장이나 낱말, 숫자, 문자열

- 인덱스와 콜론을 사용하여 문자열의 특정 부분만 잘라낼 수 잇음
- 슬라이싱을 이용하여 문자열을 나타낼 때 콜론을 기준으로 앞 인덱스에 해당하는 문자는 포함되지만 뒤 인덱스에 해당 문자는 미포함

딕셔너리(Dictionary)

- 키-값(key-value) 쌍으로 이뤄진 자료형
- Dictionary의 키(key)
  - key는 변경 불가능한 데이터(immutable)만 활용 가능
    - string, integer, float, boolean, tuple, range
- 각 키의 값(values)
  - 어떠한 형태든 관계없음
- 중괄호({}) 혹은 dict()을 통해 생성
- key를 통해 value에 접근

```
phone = {'name' : '홍길동', 'phone' : '010-1234-6789'}
print(phone['name']) # 홍길동
print(phone.get('aaa')) # None

print(phone['aaa'])
'''
in <module>
    print(phone['aaa'])
KeyError: 'aaa'
'''
print(phone.__getitem__('aaa'))
'''
in <module>
    print(phone.__getitem__('aaa'))
KeyError: 'aaa'
'''

phone.__setitem__('name', '김남길')
print(phone) # {'name': '김남길', 'phone': '010-1234-6789'}
```

형변환 (Typecasting)

1. 형변환이란?

- 데이터 형태는 서로 변환할 수 있음
- 암시적 형 변환(Implicit)
  → 자동으로 자료형을 변환
  [ex) 3/4(정수) == 0.7(실수), 0.5(실수) + 3(정수) == 3.5(실수), True(bool) + 3(정수) == 4(정수)]
  - 사용자가 의도하지 않고, 파이썬 내부적으로 자료형을 변환하는 경우
- 명시적 형 변환(Explicit)

  → int . float

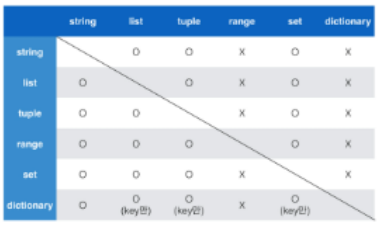
```
print('3.5'.isdigit()) #False
print('3.5'.isdecimal()) #False
print(int('0xff',16)) # 255
print(int('0x10',2)) # 2
print(int('110',4)) # 20
```

```
print(int(True)) # 1
print(int(False)) # 0
print(eval('10 * 3 + 2')) # 32
```

→ str

```
# 내부적으로 .__str__와 .__repr__가 구현되어 있다.
print([1,2,3].__str__()) # [1, 2, 3]
print([1,2,3].__repr__()) # [1, 2, 3]
```

- 사용자가 특정 함수를 활용하여 의도적으로 자료형을 변환하는 경우



	string	list	tuple	range	set	dictionary
string		O	O	X	O	X
list	O		O	X	O	X
tuple	O	O		X	O	X
range	O	O	O		O	X
set	O	O	O	X		X
dictionary	O	O (key만)	O (key값)	X	O (key값)	

map

map(함수 , 순회가능한 자료)

list(map(func, [10, 20, 30, 40]))

[func(10), func(20), func(30), func(40)]

== 반환된는 값은 자료의 요소에 한번씩 함수를 적용한 결과값을 반환한다.

```
n = input()
print(sum(list(map(int, n))))
```

set

```
digit2 = set(range(2, 1000, 2))
digit7 = set(range(7, 1000, 7))

# 합집합
print(digit2 | digit7)
# 차집합
print(digit2 - digit7)
# 교집합
print(digit2 & digit7)
#  대칭 차집합
print(digit2 ^ digit7)
```