CHAPTER_1_VARIABLES AND FUNCTIONS

01 Variables

1-(1) Python은 코드를 위에서 부터 아래로 읽는다.

```
a = 2
b = 3
c = a + b
print(c)
```

1-(2) 공백 표시는 _ 기호를 통해 표시한다.

```
my_age = 88
```

> myName 이라고 작성해도 큰 문제는 없지만 python에서는 일반적인 관습이 아니다

1-(3) 시작은 항상 글자로 시작해야 한다.

```
your_name = "pubao"
```

1-(4) 변수는 값을 저장하게 해주고 값들에 이름을 붙여줄 수 있다.

```
변수_이름 = 변수에_저장할_값
```

1-(5) 변수는 왜 필요할까?

> 프로그램에 데이터를 넣어야 하기 때문에 변수가 필요하다.
>
> 또한 다음 코드처럼 원하는 만큼 데이터를 재사용할 수 있다.

```
your_name = "wow"
your_age = 33

print("hello my name is", my_name)
print("and i'm", my_age, "years old")
```

02 Booleans and Strings

2-(1) 숫자와 문자

> 만약 숫자를 넣고 싶다면 그냥 12 이런식으로 정의할 수 있다.
>
> ```
> number = 12
> ```
>
> 그러나 문자를 넣고 싶다면 우리가 평소 글 작성할때 인뭉릐 대사를 쌍따옴표("") 처리 하는 것처럼 쌍따옴표를 붙여주면 된다. (작은 따옴표(')나 큰 따옴표(")를 사용)
>
> ```
> string = "yoyoyo"
> ```

2-(2) True와 False

> 변수에 문자, 숫자를 저장할 수 있고 True와 False와 같은 상태도 저장할 수 있다.

> true는 이진법으로 변환하면 1이다.
>
> 1을 물리적인 단어로 설명하면 on으로 설명할 수 있다 (켜진 상태)
>
> 반대로 false는 0과 off를 뜻한다.(회로에 에너지가 없다는 것처럼 0을 나타냄)

> True와 False 는 `쌍따옴표("")` 없이 사용하기

```
your_dead = False
your_dead = True
```

03 Functions

> - Functions는 코드 조각 (한번 작성한것으로 계속해서 다시 사용이 가능)

> EX) 가장 많이 사용하는 `print()` 함수
>
> : print는 값들을 console에 나타나게 해준다.

```
print("퇴근하고싶다")
```

> - Functions은 def라는 단어를 이용하여 정의할 수 있다.
>
>> `def` : "정의하다"라는 의미를 가진 영단어 define에서 나온 것이다.
>>
>> `function` : 기능, 작동(기능)하다 라는 뜻을 가졌다.

```
def say_hello():
  print("hello how r u?")

#괄호는 function 안의 코드를 실행하기 위한 것이다.
say_hello()
```

04 Indetation

> 파이썬에서 빈 공백은 코드에 영향을 끼치는 아주 중요한 요소이다 (중괄호 역활)
>
> 다른 프로그래밍 언어들은 공백을 신경쓰지 않는다.
>
> 들여쓰기로 영역 설정 및 가독성을 높인다.

```
def say_bye():
  print("bye bye")

say_bye()
```

05 Parameters

🤔 출력 결과를 수정하거나 바꾸기 위해 `function`에 어떻게 데이터를 보내면 좋을까

> - function을 만들때 데이터가 들어갈 수 있는 공간을 제공하기

```
def say_myname(user_name):
  print("hello", user_name, "how are you!!?")


say_myname("sususu")
say_myname("jake")
say_myname("pin")
```

> - `user_name` 이라는 placeholder는 `function` 안에서 사용할 수 있는 변수이다.
> - 여기서 `user_name` 은 `parameter` 이라고 한다.
> - 그리고 sususu, jake, pin 과 같은 것들은 `argument` 라고 한다.
> - 즉, placeholder는 `parameter`이고 실제로 전달한 데이터는 `argument`라고 한다.

06. Multiple Parameters

> 데이터를 넣는 순서도 중요하다.

```
def say_goodbye(user_name, user_age):
  print("goodbye", user_name, "See you later")
  print("you are", user_age, "years old")

say_goodbye("pubao", 12)
```

> 첫번째 `argument`, `"pubao"` 는 첫번째 `parameter`인 `user_name`에 저장됨
>
> 그리고 `12`는 두번째 `parameter`인 `user_age`에 저장됨

🤔 print함수에 몇개의 argument를 줄 수 있을까?

> 기억해야 할 점은 2개의 `paramenter` 가 있으면 `function`을 call 할때 2개의 `argument`을 보내야 한다. (=> 필요한 data를 전부 보내야 하기 때문이다.)

Recap

- `함수(function)`란 내가 몇번이고 재사용이 가능한 코드이다.
- `parameter`란 함수 안으로 데이터를 보내 함수의 결과를 바꿀 수 있게 해주는 것이다.
- 함수를 작성하려면`def` 라는 키워드를 작성해야 한다.
- 함수에 매번 다른 값을 보낼 수 있으면 더 좋다 이때 `parameter`가 필요한 것이다.
- `parameter`는 우리가 프로그램과 소통하기 위한 수단이다. (함수 외부로부터 받게 되는 데이터)
- 내가 원하는 데이터를 보낼 수 있도록 해당 데이터를 위한 공간을 만들어 줘야 한다.
  - 그 공간은 함수 선언문의 괄호 안이다.
  - `tex_calulator()`의 괄호 안이 바로 `placeholder`를 작성하는 곳이다.

```
def tex_calulator( ):
  print(150000 * 0.35)

tex_calulator()
```

  - 훨씬 유용한 `함수(function)`

```
def tex_calulator(money):
  print(money * 0.35)


tex_calulator(150000)
tex_calulator(120000)
```

---

07 Default Parameters

```
def anybody(user_name="anonymous"):
  print("hello", user_name)

anybody("pubao")
anybody()
```

> `anybody()`이렇게 user가 이름을 넣어주지 않는다면, `hello anoymous` 라고 해주고 싶다면,`user_name` `parameter`에 기본값을 설정하기

> `hello anonymous`라고 출력이 되는 이유는 `parameter`에 기본값을 주었기 때문이다.

> `anybody 함수`는 1개의 `rgument`를 받지만 만약 아무것도 받지 않는다면 에러를 보여주지 말고 대신에 `user_name`을 `anonymous`로 설정해달라고 하는 코드이다.

08. Return Values

- `함수(function)` 를 주스 만드는 기계라고 생각해보기

  (기계에 과일을 넣으면 출구로 나온다)

- `return` 이란 함수 바깥으로 값을 보낸다는 뜻이다.
  - 그리고 그 값을 `to_pay` 변수 안에 넣는다 그리고 나서 그 `to_pay` 변수를 가지고 `pay_tax` 함수를 부른다.
  - `to_pay` 변수에 대해 기억해야 할 건, 그게 `tax_calc 함수` 로부터 `return` 받은 값이라는 것

```
def tax_calc(money):
  # (4) return을 사용하면서 함수로부터 값을 받아낼 수 있기 때문이다.
  return money * 0.35


def pay_tax(tax):
  # (3) pay_tax 함수가 "얼마를 세금으로 내서 감사합니다." 라는 문구를 보여줄것이다.
  print("thank you for paying", tax)
  # (5) print는 값을 콘솔에서 확인할 때 매우 유용하다.


# to_pay 라는 변수를 추가함 , 이건 tax_calc 함수의 return 값과 동일하다
# (1) tax_calc(1500000) 값을 호출하면 우리에게 값을 줄것이고
to_pay = tax_calc(1500000)
# (2) 그 값을 pay_tax 호출하는데 사용할 것이다.
pay_tax(to_pay)

#------------------------------------------------------------------------

# 코드 짧은 ver
pay_tax(tax_calc(1500000))
```

Recap

(1) 변수를 포함하여 string을 만드는 방법 (format과 쌍따옴표)

> 변수를 넣고 싶은 곳에는 `{넣고 싶은 변수 이름}` 하면 끝이다.

```
name = "su kyung"
age = 29
color_eyes = "brown"

print(
    f"Hello I'm {name}, I have {age} yers in the earth, {color_eyes} is my eye color"
)
```

(2) 주스 만들기

> - juice랑 fruit 이라는 두 키워드 아주 중요
> - 기억 : return은 함수 바깥으로 값을 보내준다, python은 그 값을 가져가서 내가 작성한 코드 상에 대입해주는 것이다.
> - 중요 : return을 작성하면 python은 해당 줄 이후의 코드를 실행시키지 않는다

(1) 주스를 만드는 함수

`fruit`라는 `parameter`가 있다, `fruit`과 주스 이모티콘을 return 해준다.

(2) 얼음을 추가하는 함수

`juice`를 받아서 `juice`와 함께 얼음 이모티콘을 return 해준다.

(3) 설탕을 추가하는 함수

`iced_juice` 를 받아서 `iced_juice` 와 함께 설탕 이모티콘을 return 해준다.

```
def make_juice(fruit):
  return f"{fruit}+🥤"

def add_ice(juice):
  return f"{juice}+🧊"

def add_suger(iced_juice):
  return f"{iced_juice}+🧷"

# 1-1 : juice는 make_juice 함수의 return 값과 동일한 변수이다.
# 딸기를 make_juice라고 부른다.
juice = make_juice("🍓")
# print(juice)

# 1-2 : add_ice 함수는 우리 주스에 얼음을 추가해 return 해준다.
cold_juice = add_ice(juice)
# print(cold_juice)

# 1-3 : 과일, 주스, 얼음을 포함한 cold_juice 변수를 add_suger함수에 보내준다
perfect_juice = add_suger(cold_juice)

print(perfect_juice)
```