Python 2주차

<파일 입출력 함수와 함수 만들기>

[1. 파일 입출력 함수]

파일 입출력 함수는 말이 좀 어려워 보이겠지만, 사실 어어어엄청 쉽습니다. 우리가 여태 기본 입출력 함수로 써오던 print() 함수와 input() 함수 기억나시나요. 이 기본 입출력 함수 중

print()는 출력의 대상이 : 모니터!

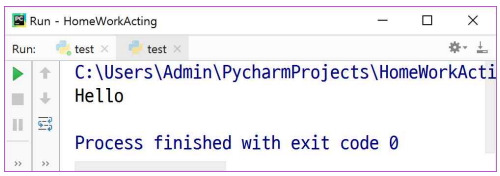
input()은 입력의 대상이 : 키보드!

입니다.

파일 입출력함수는 이 입출력 대상을 그저 "파일"로 바꾼 것 뿐입니다. 그러니까 뭔가를 출력할 때 모니터로 출력하는게 아니라 "파일"에다가 출력합니다. 입력할 때도 키보드로부터 입력을 받는게 아니라 "파일"로부터 입력을 받습니다. 다시 한번 설명해볼게요.

<기본 출력 함수 / 파일 출력 함수>

print("Hello") ==>



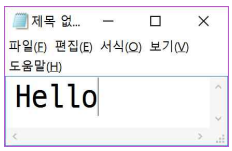
: (모니터에 출력)

파일출력함수("Hello") ==>

: (파일에 출력)

<기본 입력 함수 / 파일 입력 함수>

==> input() : (키보드로 친 문자열을 읽어들임)



==> 파일입력함수() : (파일 안의 문자열을 읽어들임)

[5-1. 파일 열고 닫기]

파일에다가 뭘 출력하려하거나 입력받으려고 하면 우선 파일을 먼저 열어야 합니다. 파일을 여는 함수는 이렇게 생겼습니다.

```
f = open("test.txt", "w")
```

open() 함수의 첫번째 인자로는 파일 이름을 써넣어줍니다.
두 번째 인자로는 이 파일을 어떤 모드로 열 것이지를 알려줘야 합니다.
이렇게 파일을 열면 이 파일 자체를 변수에다가 저장해야합니다. 위에서는 f 라는 변수에 저장한 모습입니다.

두번째 인자로 파일을 "어떤 모드"로 연다고 했는데, 이 모드에는 3가지가 있습니다.

1. "w" : 쓰기 모드

- 쓰기모드로 열면 파일 출력함수를 쓸수 있게 됩니다. 쓰기모드로 열고서 파일 읽기모드를 사용하면 에러가 납니다.

2. "a" : 추가로 쓰기 모드

- 추가로 쓰기 모드를 열면 똑같이 파일 출력함수를 쓸 수 있게 됩니다. "w"모드와의 차이점은 "w"모드는 파일 출력함수를 사용했을 때, 그 파일에 어떤 내용이 있던 간에 맨 처음부터 덮어쓰게 됩니다. 즉, 원래 있던 내용들은 모두 지워지고, 다시 새로운 내용으로 쓰여지는 거죠. 하지만 "a" 모드를 사용하면 파일 출력함수를 사용했을 때, 원래 있던 내용은 유지하면서 원래 내용의 뒷 부분부터 새로운 내용이 쓰여지게 됩니다.

3. "r" : 읽기 모드

- 읽기모드로 열면 파일 입력함수를 쓸 수 있게 됩니다. 읽기모드로 열고서 파일 출력함수를 사용하면 에러가 납니다.

파일을 열고, 원하는 만큼 다썼다면 파일을 닫아줘야 합니다. 파일을 닫는 함수는 이렇게 생겼습니다.

```
f.close()
```

열었던 파일이 f라는 변수에 저장돼 있었던 거 기억나시죠. 따라서 파일을 닫으려면 이 변수 뒤에 마침표(.)를 찍고 close() 함수를 써줘야합니다.

[5-2. 파일 출력 함수]

파일에 어떤 내용을 쓰고자 할 때, 파일 출력 함수를 사용하면 됩니다.

```
f = open("test.txt", "w")
f.write("Hello, World!")
f.close()
```

“w” 모드로 파일을 열 때, 혹시 해당 파일의 이름이 존재하지 않는다면? 그러면 에러를 뱉는게 아니라 새로운 파일을 하나 만듭니다. 이는 “a”모드도 마찬가지 입니다.

파일 출력 함수는 그 파일이 저장돼있는 변수를 명시하면서 write() 함수를 써주면 됩니다. 이렇게요

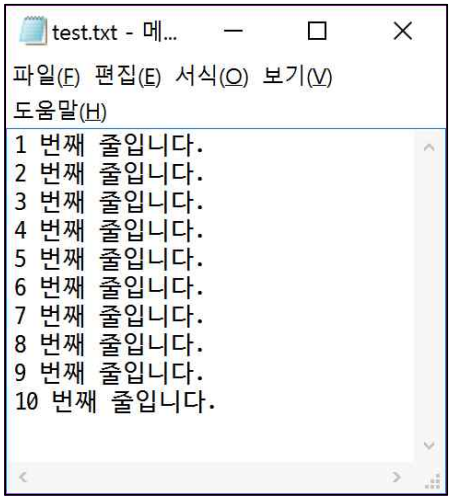
f.write() <== f 변수에는 파일이 통째로 들어가 있는 변수 입니다.

이 write() 함수는 print() 함수와 사용법이 매우 비슷합니다. 괄호 안에 내가 출력하고 싶은 것들을 넣으면 됩니다.

다만 print() 함수와 다른 것 하나는! print는 출력 후에 자동으로 “개행”을 해줬잖아요. 그치만 write() 함수는 개행을 해주지 않습니다. 출력하고 싶은 문자열의 끝에 엔터를 치고 싶다면 “\n” 을 넣어주면 되겠죠?

[5-2. 파일 입력 함수]

현재 파일에 이런 내용이 저장돼있다고 생각할게요.



이 파일의 내용을 읽어들여서 모니터(파이참)에 출력해보도록 하겠습니다.
파일 입력 함수에는 대표적으로 3가지 함수가 있습니다.

f.readline()
f.readlines()
f.read()
(=> 여기서 f는 파일이 통째로 들어가 있는 변수이름입니다.)
이 세가지 함수들을 모두 사용하여 실행해볼게요

1. readline()

```
f = open("test.txt", "r")
string = f.readline()
f.close()
print(string)
```

```
1 번째 줄입니다.


Process finished with exit code 0
```

- 메모장의 한 줄만 읽어들인 모습입니다.
  readline() 함수는 파일의 한줄씩만을 읽어들이는 함수입니다.
  두번째줄을 읽고 싶다면, readline() 함수를 한 번 더 사용하면 됩니다.

2. readlines()

```
f = open("test.txt", "r")
string = f.readlines()
f.close()
print(string)
```

```
['1 번째 줄입니다.\n', '2 번째 줄입니다.\n', '3 번째 줄입니다.\n', '4 번째 줄입니다.\n', '5 번째 줄입니다.\n', '6 번째 줄입니다.\n', '7 번째 줄입니다.\n',
'8 번째 줄입니다.\n', '9 번째 줄입니다.\n', '10 번째 줄입니다.\n']
```

- 메모장의 각 줄이 리스트형태로 저장되어 출력된 모습입니다.
  readlines() 함수는 리스트 하나를 생성하여 그 리스트에 파일의 각 줄을 값으로 저장하는 함수입니다.

3. read()

```
f = open("test.txt", "r")
string = f.read()
f.close()
print(string)
```

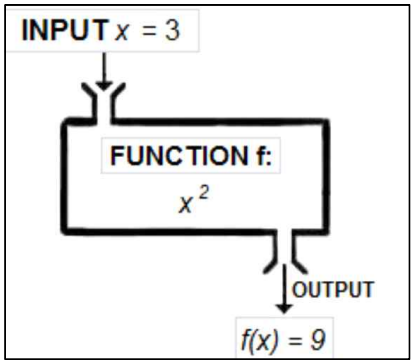
```
1 번째 줄입니다.
2 번째 줄입니다.
3 번째 줄입니다.
4 번째 줄입니다.
5 번째 줄입니다.
6 번째 줄입니다.
7 번째 줄입니다.
8 번째 줄입니다.
9 번째 줄입니다.
10 번째 줄입니다.
```

- 메모장 전체가 모두 출력된 모습입니다.
  read() 함수는 파일의 모든 내용을 “하나의 문자열”로 읽어들이는 함수입니다.

** 여기서 조금 주의할 것이, 읽기모드로 파일을 열 때, 그 파일이름이 존재하지 않는다면 에러를 내뱉는다는 것입니다. 꼭! 컴퓨터에 이미 존재하고 있는 파일이름을 사용해야합니다.

<사용자 정의 함수 만들기>

[1. 함수 기본 개념]



함수의 기본 개념은 이 그림 하나가 모두 말해주고 있어요. “입력”으로 어떤 값이 들어오면 “상자”안에서 어떤 연산을 거치고, 그 연산 결과가 “출력”되는 형태에요. 프로그래밍에서의 함수는 이 틀을 벗어나지 않습니다. 꼭 이 그림을 기억해주세요..!

[2. 파이썬 함수의 구조]

위 그림을 파이썬에서 표현해보면 다음과 같습니다.

```
def square(x):
    y = x*x
    return y
```

다시 한번 함수의 형식을 뜯어보면,

```
def 함수명(매개변수):
    <수행할 문장1>
    <수행할 문장2>
    <수행할 문장3>...
    return 반환값
```

“매개 변수”라고 써있는 부분에 “입력”으로 넣을 것들을,
“반환 값”이라고 써있는 부분에 “출력”으로 내보낼 것들을 써넣으면 됩니

다. 수행할 문장들은 꼭 들여쓰기로 작성하셔야 합니다.
함수를 만들고서 이 함수를 사용하고 싶다면, 함수 바깥에서 다음과 같이 사용하면 됩니다.(정의된 함수를 사용하는 것을 보고, 함수를 "호출"한다고 말합니다.)

```
>>>  result = square(3)
```

square 함수의 입력값으로 3을 넣었고, 이 3 이라는 수는 함수 내부의 변수인 x 에 전달되어, 함수 내부에서는 제곱 연산을 거친 후, 결과 값으로 9를 내보내게 됩니다. 이 9 라는 수는 result라는 변수에 저장됩니다.

다음은 세 개의 수를 더해주는 함수를 정의하고, 이 함수를 호출해 본 모습입니다.

```
def sum_three(a,b,c):
    y = a+b+c
    return y

result = sum_three(1,2,3)
print("결과 :", result)
```

[3. 함수의 다양한 형태]

함수는 입력과 출력이 있다고 했는데, 입력과 출력이 없어도 함수를 정의할 수 있습니다. 아래부터는 함수의 다양한 형태를 보여줍니다.

[3-1. 입력이 없는 함수]

```
def STRING():
    str = "Hello, World!"
    return str

result = STRING()
print(result)
```

입력으로 들어가는 변수가 없고, 함수의 끝에 return 만 있는 형태입니다.

함수 호출할 때도 보시면 입력값으로 아무것도 안 넣은 것을 확인할 수 있습니다.

[3-2. 출력이 없는 함수]

```
def PRINT(str):
    str_plus = "결과 :"+str
    print(str_plus)

PRINT("Hello, World!")
```

입력으로 들어가는 변수만 있지만, return 이 없는(출력이 없는) 형태의 함수입니다. 함수 호출시에도 출력으로 나오는 값이 없기 때문에 반환값(출력값)을 받아줄 변수를 사용하지 않은 것을 볼 수 있습니다.

[3-3. 입력, 출력이 없는 함수]

```
def PRINT_HELLO():
    print("Hello World!")

PRINT_HELLO()
```

입력으로 들어가는 변수도 없고, return으로 반환되는 값도 없는 형태입니다. 함수 호출시에도 보면, 입력값과 반환값이 없기 때문에 입력으로 아무것도 넣지 않았고, 반환값을 받아줄 변수도 사용하지 않았습니다.
함수를 호출 하면 그저 함수 내부의 문장들만 수행합니다.

[3-3. 출력이 여러개인 함수]

출력값으로 하나가 아닌 두 개 이상을 내보낸다면 어떻게 될까요? 두 개의 수를 입력으로 받고 이 두 수를 1씩 증가시킨 것을 반환하는 함수를 만들어 보겠습니다.

```
def increase(a,b):
    return a+1, b+1

result = increase(10, 20)
print(result)
```

increase 라는 함수 내부를 보면, return 값으로 두개를 적은 것을 봐주세

요. 이를 호출했을 때, 반환 값은 어떤 형태로 반환이 되는지 확인해보겠습니다.

```
(11, 21)

Process finished with exit code 0
```

result 를 출력했더니 소괄호 안에 값들이 들어있는 것을 볼 수 있습니다. 즉, 값을 두개 이상을 한번에 반환하면 그 반환값은 “튜플”형으로 반환됩니다.

[4. 매개변수에 초기값 미리 설정하기]

```
def forecast(temp, wind, weather):
    print("온도 : {}도".format(temp))
    print("바람 : {}m/s".format(wind))
    print("날씨 : {}".format(weather))
```

다음과 같이 기온, 바람, 날씨를 입력해주면, 정해진 서식으로 이쁘게 출력해주는 함수가 있습니다. 그러면 함수 호출은 아래와 같이 할 수 있습니다.

>>> forecast(23, 1, “맑음”)

그런데 이 함수에서 weather 변수에 초기값을 “맑음”으로 설정해준다면 함수 호출시 더 간단해집니다.

```
def forecast(temp, wind, weather="맑음"):
    print("온도 : {}도".format(temp))
    print("바람 : {}m/s".format(wind))
    print("날씨 : {}".format(weather))

forecast(23, 1)
```

함수 호출 부분을 주목해주세요. 원래는 입력으로 3개의 값을 입력해줘야 하는데 두 개만 넣어줬습니다. 이게 가능한 이유는 함수를 정의할 때 매개변수의 초기값을 미리 설정해줬기 때문인데요. 함수 정의 부분의 매개변수 부분을 보면..

weather=“맑음”

이라고 써진 부분이 바로 초기값을 설정해준 부분입니다. 이렇게 초기값을 설정해주면 함수 호출시 해당 값을 안 써넣어줬을 때 자동으로 매개변수에

는 이 초기값이 들어가게 됩니다.
이 함수 초기값을 설정할 때 주의할 점은 꼭! 매개변수들의 마지막 순서에 와야한다는 것입니다. 아래와 같이 초기값이 설정된 매개 변수를 중간 위치에 넣으면 에러가 납니다.

```
def forecast(temp, weather="맑음", wind ):    # 에러!!
    print("온도 : {}도".format(temp))
    print("바람 : {}m/s".format(wind))
    print("날씨 : {}".format(weather))

forecast(23, 1)
```

[5. 지역 변수, 전역 변수]

```
def function():
    a = 1
    return a
```

위와 같은 함수가 있고, 함수 바깥에서 이 함수 안에 있는 변수인 a에 한번 접근 해보겠습니다.

```
def function():
    a = 1
    return a

print(a)
```

```
Traceback (most recent call last):
  File "C:/Users/Admin/PycharmProjects/Study/test.py", line 5, in <module>
    print(a)
NameError: name 'a' is not defined

Process finished with exit code 1
```

함수 바깥에서 함수 안의 변수에 접근하려고 하니, 에러가 납니다. 이 이유는 함수 안에 있는 변수는 함수가 “호출”될 때만 생성됐다가, 호출이 끝날 때 소멸되기 때문입니다.

```
def function():
    a = 1
    return a

function()
print(a)
```

```
Traceback (most recent call last):
  File "C:/Users/Admin/PycharmProjects/Study/test.py", line 6, in <module>
    print(a)
NameError: name 'a' is not defined

Process finished with exit code 1
```

함수를 호출하여 변수 a를 생성시켰는데도, 변수 a에 접근하려 하니 에러가 나는 것을 보니 변수가 소멸됐다는 것을 확인할 수 있습니다.
이렇게 함수 안에 있는 변수를 보고 “지역변수”라고 합니다. 지역변수는 무조건 그 변수가 속해있는 함수 안에서만 접근(참조)가 가능한 것이죠.
“전역 변수”는 반대로 함수를 포함한 모든 공간에서 접근(참조)가 가능합니다. 하지만 파이썬에서의 전역 변수는 조금 특이합니다. 우선 함수에서 전역 변수에 접근하는 모습을 보여드리겠습니다.

```
num = 10

def function():
    a = num
    print(a)


function()
```

```
10

Process finished with exit code 0
```

num이라는 변수는 함수 바깥에 선언이 돼있어서 “전역 변수”로서 작동합니다. 따라서 num은 모든 공간에서 접근할 수 있기 때문에, function() 함수 내부에서 에러 없이 정상적으로 가져다 사용한 모습을 볼 수 있습니다.
하지만 아래의 예제를 볼게요.

```
num = 10

def function():
    num = num+1
    print(num)


function()
```

```
Traceback (most recent call last):
  File "C:/Users/Admin/PycharmProjects/Study/test.py", line 8, in <module>
    function()
  File "C:/Users/Admin/PycharmProjects/Study/test.py", line 4, in function
    num = num+1
UnboundLocalError: local variable 'num' referenced before assignment

Process finished with exit code 1
```

함수 내부에서 전역변수를 가져다 쓴 것 뿐인데, 이번에는 에러가 납니다. 이 이유는 전역변수를 가져다가 *함부로* 값을 수정하려 했기 때문입니다. 즉 파이썬에서는 전역변수를 사용할 땐 무조건 read-only 여야 합니다. 그렇다면 정말로 전역변수의 값을 함부로 수정하지 못하는 걸까요? 이를 위한 해결책이 존재합니다.

```
num = 10

def function():
    global num
    num = num+1
    print(num)


function()
```

```
11

Process finished with exit code 0
```

함수 내부에서 전역변수에 접근하고자 할때,

    global (변수이름)

을 써주시면 됩니다. 이를 써주게 되면, 인터프리터에게

    “나 전역변수 쓸건데 그 전역변수 값을 수정할거야. 알고 있어”

라고 말해주게 됩니다.

[6. lambda : 함수 더 간단하게 쓰기]

이제 덧셈 기능을 하는 함수를 만들고, 그 함수를 호출하여 두 수의 합을 구하는 것은 눈 감고도 할 수 있을 것이라 생각이 듭니다. 한번 만들어볼게요.

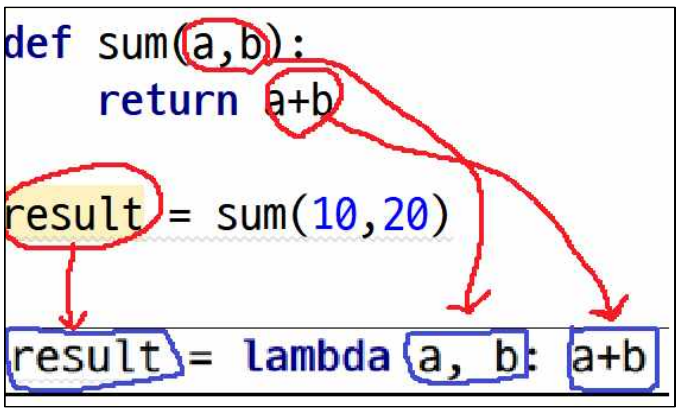
```
def sum(a,b):
    return a+b

result = sum(10,20)
```

눈 감고 했습니다..
자 그런데 저 3줄짜리 코드를 한 줄로 나타내는 방법이 있습니다!

```
result = lambda a, b: a+b
```

“lambda” 라는 명령어를 쓰면 함수의 정의와 호출을 한 줄로 한번에 나타낼 수 있습니다! 안 그래도 쉬운데, 쉬운걸 더 쉽게 만드는 파이썬.. 당신은 도대체..
lambda를 사용하는 방법은 다음과 같습니다.



음.. 제 설명이 좀 불친절했나요..ㅠㅠ 이것보다 더 친절할 수 없을 것같은데.. 이해 못하시겠으면 얼른 손가락을 움직이세요! 프로그래밍 언어는 손가락을 많이 움직이는 사람이 더 많이 배웁니다. ㄹㅇ..