

## 부록

## 파이썬 병아리반

DATA ANALYSIS FOR EVERYONE



부록에서는 파이썬 기초를 빠르게 훑어 보고 지나가겠습니다. 따라서 개념 부분에 대한 설명은 자세하게 하지 않고, 코드를 중심으로 설명합니다. 이 책에서 다루는 기초 문법(입력, 출력, 연산자, for 반복문, if 조건문, 리스트 등)을 가볍게 복습하는 차원에서 또는 파이썬을 처음 배우는 사람들이 맛보기로 읽어도 좋습니다.

만약 다음 코드가 어렵지 않은 사람은 부록을 따로 읽지 않아도 괜찮습니다.

```
names = ['초파', '루피', '상디', '조로']
names.append('해적왕')
for name in names :
    if len(name) > 2 :
        print(name, '왔나요~?')
```



### 1

### 출력과 입력 그리고 변수

지금부터 이 책에서 다루지는 파이썬의 가장 기초적인 내용을 살펴보려고 합니다. 먼저 여러분이 파이썬을 설치했다고 가정하고 진행하겠습니다.

## ■ print() 함수로 출력하기

여러분은 아마 print라는 단어가 친숙할 겁니다. 흔히 ‘프린트 했어?’ 이런 표현들을 일상 생활에서도 많이 사용하는데요.

파이썬에서 print라는 이름의 함수는 괄호 안의 어떤 값을 모니터에 출력할 때 사용합니다. 백문이 불여일타(百聞而不如一打)라고 했으니 먼저 코드를 작성해 보겠습니다. 자, print는 괄호 안의 어떤 값을 모니터에 출력해줄 때 사용한다고 했는데요, 파이썬을 처음 배우더라도 이 코드가 무엇을 의미하는지 짐작해 보세요.

```
print(34759**24341)
```

어떤 코드처럼 보이나요?



음..곱하기가 두 개니까.. 혹시 제공하는 코드인가요?

네, 맞습니다. 이 코드를 실행하면 무려 110,535 자리의 큰 수가 출력됩니다.




그러면 print() 함수는 숫자만 출력할 수 있나요?

아닙니다. 코드를 조금 수정해서 다음과 같이 작성하면 텍스트도 출력할 수 있습니다. 보통 프로그램에서 텍스트를 ‘문자열’이라고 표현하는 경우가 많고, 영어로는 ‘string’이라고 표현한다는 점을 알아두기 바랍니다.

```
print('hello world')
```

실행  
결과

```
hello world
```

 그러면 괄호 안의 작은따옴표(' ') 안에 원하는 텍스트를 넣으면 그 텍스트가 출력되는 건가요?


네, 맞습니다. 작은 따옴표 안에 값을 바꿔서 원하는 텍스트를 출력하세요.

## ■ 변수

변하는 값, 즉 변수에 대해 알아보겠습니다. 여러분도 웹 사이트에 로그인했을 때 이런 글귀를 본 적이 있을 겁니다.

파이썬님! 안녕하세요!

그런데 이 글귀에서 어떤 부분은 사람에 따라 바뀌고, 어떤 부분은 사람에 관계 없이 바뀌지 않는다는 것을 쉽게 알 수 있습니다. 어디가 바뀌는 부분인가요?

 ‘파이썬’이라는 닉네임 부분이 바뀌겠네요.

네, 맞습니다. 컴퓨터에서는 데이터를 저장하기 위해 공간이 필요합니다. 닉네임 부분처럼 하나의 값으로 고정되어 있지 않고 변하는 값을 저장할 수 있는 공간을 ‘변수(變數, variable)’라고 합니다.

변수는 다음과 같이 사용할 수 있습니다.

```
name = '파이썬'  
print(name)
```

예상하는 것처럼 ‘파이썬’이라는 값에 원하는 값을 넣으면 다양한 값이 출력되는 것을 볼 수 있으니 값을 다양하게 바꿔보세요.

'name = '파이썬'' 코드에서 name이 바로 변수이고 = 기호는 오른쪽의 값을 왼쪽에 대입(또는 배정)하는 역할을 합니다. 값을 대입한다고 해서 대입 연산이라고도 불러요. 수학에서는 같다는 의미로 사용해서 조금 헷갈릴 수도 있지만, 자주 볼 테니 곧 익숙해질 겁니다.



그러면 변수의 이름은 반드시 name이라고 써야 하나요? 다른 이름으로 할 수도 있나요?

좋은 질문입니다. '변수의 이름은 숫자로 시작하면 안 된다' 같은 몇 가지 제한적인 규칙을 제외하면 여러분이 원하는 대로 써도 됩니다. 그리고 = 기호 오른쪽에 있는 값은 문자열이어도 되고, 숫자도 상관없습니다. 단, 변수가 어떤 값을 저장하는지를 잘 나타낼 수 있는 이름을 정하는 것이 바람직하다는 점은 꼭 기억하세요!

```
a = 1024 # 변수의 이름은 여러분이 지정하면 됩니다.  
print(a)
```

이제부터는 어떤 궁금증이 생겼을 때는 직접 테스트해 볼 수 있는 코드를 작성하거나 검색을 하면서 스스로 해결하면 성취감도 높아지고, 실력도 많이 늘 겁니다. 변수는 다음과 같이 사용할 수 있습니다.

```
# name 변수의 값을 '파이썬' 대신 다른 값으로 바꿔보세요!  
name = '파이썬'  
print(name+'님! 안녕하세요!')
```

실행  
결과

```
파이썬님 안녕하세요!
```

여기에서 두 가지 사실을 알 수 있는데요. + 기호가 문자열과 문자열 사이에 있을 때는 두 문자열을 연결하는 접착제 같은 역할을 한다는 사실과, #으로 시작하는 문장은 코드의 실행 결과에 영향을 주지 않는다는 사실입니다. 이렇게 #으로 시작하는 문장을 주석(comment)이라고 합니다. 코드에 대한 설명 글을 적을 때 주로 주석을 사용합니다.

```
name = '원더키디' # 주석은 코드 오른쪽에 쓸 수도 있습니다.  
print(2020, name, '화이팅!!')
```

실행  
결과

```
2020 원더키디 화이팅!!
```

파이썬 병아리반에서는 변수에 대해 이 정도만 이해해도 될 것 같습니다. 그러면 조금 더 재미있는 입력하는 방법을 배워보겠습니다.

#### ■ input() 함수로 문자열 값 입력받기

다음 코드를 실행해보세요.

```
name = input()  
print(name+'님! 안녕하세요!')
```

실행했는데 아무런 일이 일어나지 않아서 당황했나요? 여러분의 이름을 키보드로 입력하세요.

실행  
결과

**파이썬** → 파이썬 또는 여러분의 이름을 입력하세요!  
파이썬님! 안녕하세요!

앞에서 `print()` 함수는 괄호 안의 값을 모니터에 출력해주는 역할을 하는 함수라고 했었죠?

함수(函數, function)는 함(函, 상자 함)의 뜻처럼 상자 안에 값을 넣으면 어떤 기능을 수행합니다. 파이썬에서는 함수 이름 뒤에 소괄호가 열리고 닫히는 형태로 표현됩니다. 그리고 `input`이라는 함수는 키보드로 문자열 값을 입력받는 기능을 하는 함수입니다.

그런데 어떤 값을 입력하라는 안내가 없어서 조금 불편한 것 같습니다. 그래서 코드를 조금 더 수정하겠습니다.

```
name = input('이름을 입력해주세요 : ')\nprint(name+'님! 안녕하세요!')
```

실행  
결과

이름을 입력해주세요 : **파이썬** → 파이썬 또는 여러분의 이름을 입력해 보세요!  
파이썬님! 안녕하세요!

이렇게 `input()` 함수의 괄호 안에 원하는 안내 메시지를 추가할 수도 있습니다. 여기까지 잘 이해가 되나요?



네. 그런데 왜 ‘문자열 값’을 입력받는다고 설명했나요? 숫자는 안 되나요?

예리한 질문입니다. 다음 코드를 실행하세요.

실행  
결과

```
age = input('나이를 입력해주세요! : ')
print(age-4)
```

나이를 입력해주세요! 25 → 25 또는 여러분의 나이를 숫자로 입력하세요!

```
TypeError                                Traceback (most recent call last)
<ipython-input-19-3dcec6c3c897> in <module>()
      1 age = input('나이를 입력해주세요! : ')
----> 2 print(age - 4)


TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

에러가 나왔네요! 처음에는 당황스러울 수 있지만, 사실 프로그래밍을 처음 배울 때 에러가 한 번도 나지 않는 사람은 없습니다. 병아리반처럼 아직 파이썬이 익숙하지 않은 사람들에게는 에러를 고치는 과정에서 배우는 것이 아주 중요합니다. 맨 밑의 에러 메시지를 볼까요?

```
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

참고로 `str`은 `string`, 즉 문자열의 약자이고, `int`는 `integer`, 즉 정수(整數)의 약자입니다. 즉, 이 메시지는 우리가 문자열 타입의 값에서 정수 타입의 값을 빼려고 해서 에러가 발생했다는 사실을 알려주고 있습니다. 다시 말해 25라는 숫자처럼 보이는 값을 입력했지만, 사실 `input()` 함수로 입력받은 값은 문자열 타입으로 저장되었다는 것을 알 수 있습니다.

그러면 str과 int 중 어떤 값을 어떤 타입으로 바꿔줘야 할까요?

 age에서 4를 빼려는 의도였으니, 문자열 타입인 age를 정수 타입으로 바꿔야 해요.

네, 맞습니다. 다음과 같이 int() 함수를 사용해서 정수 타입으로 바꾸면 에러를 고칠 수 있습니다.

```
age = input('나이를 입력해주세요! : ')
print(int(age) - 4)
```

실행  
결과

```
나이를 입력해주세요! : 25 → 25 또는 여러분의 나이를 숫자로 입력해 보세요!
21
```

출력, 입력, 변수에 대한 내용은 다음 코드로 정리하며 마치겠습니다. int() 함수의 위치를 잘 살펴보고, str() 함수의 역할이 무엇인지 스스로 생각해 보세요!

```
name = input('이름을 입력해주세요 : ')
age = int(input('나이를 입력해주세요 : '))
print('안녕하세요!', name+'님! 저는 처음에 '+str(age - 4)+'살인 줄  
알았어요!!')
```



## 2

## 연산자 사용하기

연산자(演算子, operator)라는 말을 처음 들으면 무척 낯설게 느껴질 텐데요, 사실 우리는 연산자를 본 적이 있습니다. 그것도 4개나 말이죠. +, -, \*\*, 그리고 오른쪽 값을 왼쪽 변수에 대입하는 역할을 했던 =도 연산자입니다. 코드를 차근 차근 따라해보며 값을 이리저리 바꾸다 보면 금세 익숙해질 겁니다.

### ■ 산술 연산자(arithmetic operator)

첫 번째 만날 연산자는 산술(算術, arithmetic) 연산자입니다. 쉽게 말해서 숫자 계산을 위한 기호라고 생각하면 됩니다. 그리고 딱 이만큼만 알면 됩니다. 값을 이렇게 저렇게 바꿔보고, 여기에 없지만 앞에서 봤던 +, - 기호를 활용해서 계산을 해보기 바랍니다.

```
# 다양한 값을 넣어보며 연습해보세요.  
print(3 * 10)    # *: 곱셈 연산자  
print(3 ** 10)   # **: 거듭 제곱 연산자  
print(3 % 10)    # %: 나머지 연산자  
print(3 / 2)     # /: 실수 나눗셈  
print(3 // 2)    # //: 정수 나눗셈
```

### 실행 결과

```
30  
59049  
3  
1.5  
1
```

모든 연산자를 외울 필요는 없습니다. 필요한 상황에서 적절히 골라 사용할 수 있도록 어떤 연산자가 있는지만 알면 됩니다. 더 이상의 설명은 생략하겠습니다.

### ■ 비교 연산자(comparison operator)

두 번째 만날 연산자는 비교 연산자입니다. 이것도 코드를 보면 직관적으로 쉽게 이해가 갈 겁니다. 그리고 비교 연산자를 활용한 식이 참이면 True, 거짓이면 False라는 값을 갖는다는 것을 기억하세요.

```
# 다양한 값을 넣어보며 연습해보세요.  
print(10 >= 3)  
print(10 <= 3)  
print(10 == 3)    # ==: 같다  
print(10 != 3)    # !=: 같지 않다  
print(3 % 2 == 1)
```

#### 실행 결과

```
True  
False  
False  
True  
True
```

여기에서는 마지막 명령만 살펴보겠습니다. 먼저 3 % 2는 결과가 무엇일까요?



3을 2로 나눈 나머지가 1이에요!

네, 맞습니다. 그러면 3 % 2는 1과 같나요?

네. 그래서  $3 \% 2 == 1$ 이 참이니까 결과가 True라고 나온 것이군요?

네, 정확합니다. 그러면 여기에서 순서가 중요할까요? 예를 들면,  $=>$ ,  $!=$  이렇게 써도 될까요?

또  $==$  이렇게 띄어쓰기를 해도 문제없이 잘 실행될까요?

음... 잘 모르겠어요!

여러분이 직접 코드로 작성해보세요. 결과를 쉽게 알 수 있을 겁니다.

#### ■ 논리 연산자(logical operator)

어렵게 느껴지던 연산자도 벌써 마지막 논리 연산자만을 남겨두고 있습니다. 논리 연산자는 앞에서 보았던 참과 거짓, 즉, True와 False와 관련된 연산입니다. 여기에서 다루지 않는 논리 연산자도 있지만, 병아리반에서는 다음의 and, or, not 3가지 연산자만 알면 충분합니다.

먼저 논리곱은 A and B와 같은 형태로 표현합니다. 다음 표에서 보는 것처럼 A와 B 중 하나라도 False일 경우에 결과는 False이고 둘 다 True일 때만 결과가 True가 됩니다.

논리식 (A and B)	결과
True and True	True
True and False	False
False and True	False
False and False	False

**표 부록-1**  
논리곱 진리표

# 다양한 값을 넣어보며 연습해보세요.

```
print(1 == 1 and 2 != 1)          # True and True
```

```
print(10 % 2 != 0 and 1 + 1 > 0)  # False and True
```

### 실행 결과

True

False

논리합은 A or B와 같은 형태로 표현합니다. 표 부록-2에서 보는 것처럼 A와 B가 모두 False일 경우에만 결과가 False이고, 나머지 경우 True입니다. 즉, A와 B 중 둘 중 하나라도 True이면 결과는 True가 됩니다.

**표 부록-2**  
논리합 진리표

논리식 (A or B)	결과
True or True	True
True or False	True
False or True	True
False or False	False

# 다양한 값을 넣어보며 연습해보세요.

```
print(10 < 5 or 10 == 5)          # False or False
```

```
print(10 % 2 != 0 or 1 + 1 > 0)    # False or True
```

실행  
결과

False  
True

논리 부정은 not A와 같은 형태로 표현합니다. 다음 표에서 보는 것처럼 A가 False일 때 결과는 True, True일 때는 False가 됩니다.

표 부록-3  
논리 부정 진리표

논리식 (not A)	결과
not True	False
not False	True

# 다양한 값을 넣어보며 연습해보세요.

```
print(not 10 > 5)    # not True
print(not 10 == 5)   # not False
print(not 0)         # 0은 False에 해당
print(not 10)        # 0을 제외한 숫자도 True에 해당
```

실행  
결과

False  
True  
True  
False

### 3

## 함수 불러오기

지금까지 우리가 사용했던 함수에는 어떤 것들이 있었나요?



print, input, int, str이요!

지금까지 사용했던 함수들은 파이썬에서 항상 사용할 수 있도록 제공하는 **내장 함수(Built-in Functions)**입니다. 현재 파이썬 3.7 버전에서 제공하는 내장 함수의 종류는 다음과 같습니다. 이중 몇 가지는 앞으로 사용하겠지만, 모두 알 필요는 전혀 없습니다.

내장 함수				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

#### 표 부록-4

파이썬 내장 함수

그런데 내장 함수 중에는 제곱근(square root)을 구할 수 있는 함수도 없고, 랜덤한 숫자를 만들어낼 수 있는 랜덤 함수도 없습니다. 그렇지만 이런 함수들은 별도로 불러서 사용할 수 있는데요. 내장 함수 외의 함수들을 불러올 때는 다음과 같이 import라는 명령을 사용합니다.

여기에서 import random은 random이라는 이름의 라이브러리를 불러오라는 명령이고, random.randint(1,6)은 random 라이브러리에 있는 randint라는 함수

에 1과 6을 입력해서 실행하라는 의미입니다. randint() 함수는 입력되는 두 정수 사이의 숫자를 랜덤하게 생성하는 함수로 random.randint(a, b)는 a 이상 b 이하의 한 개의 정수를 만듭니다.

```
# random 숫자의 범위를 바꿔보세요.  
import random  
dice = random.randint(1,6)  
print(dice)
```

실행  
결과

5

같은 방식으로 제곱근 함수를 사용하려면 math라는 수학 관련 라이브러리를 불러와서 math 라이브러리에 있는 sqrt() 함수를 실행시키면 됩니다.

```
import math  
print(math.sqrt(2))
```

실행  
결과

1.4142135623730951



그러면 내가 원하는 기능을 하는 함수가 어떤 라이브러리에 있는지 어떻게 알 수 있을까요?

그건 인터넷에서 검색합니다. 프로그래밍을 하면서 잘 모를 때는 구글에 물어보는 것이 가장 좋아요. 그리고 다음 URL의 파이썬 공식 문서에 방문해서 찾아보는 것도 좋은 습관입니다.

- [URL](https://docs.python.org/ko/3/library/index.html) <https://docs.python.org/ko/3/library/index.html>

## 4

### 반복과 선택

이제 가장 중요한 부분입니다. 컴퓨터는 사람보다 빠르게 많은 양의 데이터를 처리할 수 있습니다. 이때 반복과 선택을 통해 정확하고 효율적인 동작이 이루어질 수 있도록 프로그래밍할 수 있습니다.

#### ■ for 반복문

먼저 명령어를 반복하여 실행시키는 방법을 알아봅시다. 반복문을 적절히 사용하면 단 몇 줄의 코드로도 원하는 결과를 얻을 수 있습니다.

파이썬에서 for 반복문은 주어진 데이터 세트를 순회하거나 원하는 횟수만큼 반복하고 싶을 때 사용합니다.

for 반복문의 구조는 다음과 같습니다.

```
for A in B :           # 맨 뒤에 콜론 기호(:)를 잊지 마세요.  
    반복할 문장1       # 들여쓰기(문단 구분)가 되어 있음  
    (반복할 문장n)     # 보통 공백 문자(space) 4개로 사용함
```

여기에서 B는 데이터 세트를 의미합니다. 그리고 데이터 세트에서 데이터를 하나씩 꺼내 와서 A에 임시로 저장합니다. 그리고 데이터 세트 뒤에 “:”을 쓰고 그 밑에 반복할 문장을 작성합니다. 여기에서 반복할 문장은 문단 구분 같은 것이 되어있는 것을 볼 수 있는데요. 바로 파이썬을 처음 배울 때 가장 성가신 점인 들



여쓰기(indent) 규칙입니다. 자동으로 들여쓰기가 되는 경우도 있지만 직접 들여쓰기를 해야할 경우에는 공백 문자, 즉 `Space`를 4번 치면 됩니다.

이제 코드로 예를 들겠습니다. 다음 코드가 실행되면 '파이썹'이라는 문자열을 순회하며 한 글자씩 출력합니다. 여기에서는 '파이썹'이라는 문자열이 데이터 세트인 것이고, 이 데이터 세트에서 '파', '이', '썹'이라는 데이터를 하나씩 순서대로 꺼내서 변수 `i`에 임시로 저장합니다. 즉, 첫 번째 바퀴에서 `i`에는 '파'가, 다음 바퀴에서는 '이'가, 세 번째 바퀴에서는 '썹'이 저장됩니다.

그래서 `i`를 반복하며 출력하면 '파', '이', '썹'이라는 글자가 하나씩 출력됩니다.

```
# name 변수의 값을 바꿔보세요.  
name = '파이썹'  
for i in name :  
    print(i)
```

#### 실행 결과

```
파  
이  
썹
```

설명은 어렵지만 다음 코드의 실행 결과를 예측할 수 있을 겁니다.

```
names = ['쵸파', '루피', '상디', '조로']  
for name in names :  
    print(name)
```

어떤 결과가 나올까요?



초파, 루피, 상디, 조로라는 이름이 한 줄씩 출력될 것 같아요. names가 데이터 세트이고 여기에서 하나씩 데이터를 꺼내 와서 name이라는 변수에 저장해서 출력하기 때문이죠.

훌륭합니다. 혹시 다음 질문도 대답할 수 있으면 for 반복문에 대한 이해는 충분합니다. 다음 코드는 어떤 결과를 출력할까요?

# 다음 결과를 확인한 후, i \*\* 2 대신 다른 식을 넣어 동작을 확인하세요!

```
for i in [0,1,2,3] :
    print(i ** 2)
```



[0,1,2,3]이라는 데이터 세트에서 하나씩 데이터를 꺼내 와서 i라는 변수에 저장한 후 i의 제곱을 출력해서 0, 1, 4, 9가 한 줄씩 출력됩니다.


맞습니다. 만약 이것이 잘 이해가 안 가면 위의 코드에서 데이터 세트와 식을 바꿔가며 연습해 보세요. for 반복문은 아주 중요하기 때문입니다.

그런데 숫자의 범위를 0~3이 아닌 0~99까지로 늘리려면 [0, 1, 2, 3, ..., 99] 이렇게 코드를 작성해야 할까요? 이렇게 비효율적인 일을 하지 말라고 프로그래밍을 하는 것이기 때문에 분명 좋은 방법이 있을 겁니다. 다음과 같이 말이죠.

```
for i in range(100) : # range(100): 0 이상 100 미만(0, 1, 2, ..., 99)의 범위를 갖는 정수
    print(i ** 2)
```

여기에서 range(100)는 0부터 100미만의 연속된 정수를 생성합니다. 그러면 다음 코드의 print() 함수는 몇 번 반복이 될까요? 네, 0부터 99까지 총 100번 반복이 됩니다. for 반복문과 range() 함수를 사용하면 이렇게 원하는 횟수만큼 반복을 시킬 수도 있습니다.

```
for i in range(100) : # 0부터 99까지의 정수 100개 생성. 즉, 100번 반복
    print('나는 파이썬왕이 될 사람이다!!')
```

 그런데 꼭 정수를 0부터 생성해야 하나요? 예를 들어 1부터 9까지 반복하고 싶을 땐 어떻게 해요?

만약 1부터 9까지의 정수를 생성하고 싶으면 `range(1, 10)`이라고 코드를 작성하면 됩니다. 그리고 하나의 옵션이 더 있어서 `range(2, 100, 3)`라고 작성하면 2부터 100 미만의 정수가 3씩 간격(2, 5, 8, 11, ..., 95, 98)을 두고 생성됩니다. 자주 쓰이지 않지만 알아두면 요긴하게 쓸 일이 있을 겁니다.

## ■ if 조건문

우리는 매일 매일 수많은 선택을 하며 살아가는데요. 컴퓨터가 조건에 맞게 명령어를 선택하도록 조건문을 사용할 수 있습니다. 프로그래밍에서 선택은 반복과 함께 가장 중요한 개념입니다.


파이썬을 포함한 대부분의 프로그래밍 언어에서 if 조건문은 주어진 조건이 True(참)인 경우에만 명령을 선택적으로 실행하고 싶을 때 사용합니다.

if 조건문의 구조는 다음과 같습니다. for 반복문과 마찬가지로 콜론 기호(:)를 사용하고 조건이 참일 때 실행될 문장은 4칸 들여쓰기를 해야 합니다.

```
if 조건 :                               # 맨 뒤에 ':' 콜론을 잊지 마세요!
    조건이 참일 경우 실행할 문장1       # 들여쓰기가 되어 있음
    (조건이 참일 경우 실행할 문장1)     # 들여쓰기를 잊지 마세요!
```


코드를 보면 더 쉽게 이해할 수 있습니다. 다음 코드에서 if 뒤의 조건은 참인가요?

```
if 10 > 0 :  
    print('안녕하세요?')
```

 네, 10은 0보다 크다는 조건은 True(참)입니다. 그러면 그 다음 들여쓰기 된 print() 함수가 실행되는 건가요?

네, 정확합니다. 그러면 다음 코드는 어떻게 실행될까요?

```
if 10 != 0 and 5 % 2 == 1 :  
    print('안녕하세요?')
```

 10은 0과 같지 않다는 조건이 True고, 5를 2로 나눈 나머지가 1이라는 조건도 True니까 True and True는 True겠네요. 그러면 print() 함수가 실행됩니다.

훌륭합니다. 앞에서 우리가 논리, 비교 연산자를 배웠었는데요. 방금 본 코드에서는 != 나 ==와 같은 비교 연산, and 와 같은 논리 연산이 함께 쓰인 것을 볼 수 있습니다. 이렇게 원하는 조건식을 만들기 위해 논리, 비교, 산술 연산을 배운 것입니다.

그러면 이 코드를 읽어볼까요? 앞에서 배웠던 input() 함수, int() 함수, 변수와 함께 if문(statement)을 사용하면 이런 프로그램을 만들 수 있습니다.


```
passwd = int(input('비밀번호 4자리를 입력하세요 : '))  
if passwd == 1531 :  
    print('비밀번호가 일치합니다.')
```

그리고 이 코드를 앞에서 배웠던 for 반복문과 함께 사용하면 병아리반 수준의 해킹 프로그램을 만들 수 있습니다. 여기에서 for 반복문 안에 if가 들어쓰기 되었고, if 조건문 안의 print 구문이 들어쓰기 된 것을 눈여겨보기 바랍니다.

```
for i in range(10000) :  
    if i == 1531 :                # 1단계 들어쓰기  
        print('비밀번호가 일치합니다.')    # 2단계 들어쓰기
```

이제 기본적인 if 조건문에서 한 걸음 더 나아가보겠습니다. 여러분은 else라는 단어가 ‘그 밖에’라는 뜻을 갖고 있다는 사실을 알 겁니다. 그러면 이 코드의 밑줄 친 부분에는 어떤 메시지가 들어가면 자연스러울까요?

```
passwd = int(input('비밀번호 4자리를 입력하세요 : '))  
if passwd == 1531 :  
    print('비밀번호가 일치합니다.')  
else :  
    print('_____')
```

 조건이 참이 아닌 그 밖의 경우에는 ‘비밀번호가 일치하지 않습니다’라는 문장이 들어가면 자연스러울 것 같습니다.

맞습니다. 그래서 if와 함께 else를 사용하면 조건이 참일 때와 거짓일 때의 명령을 각각 다르게 실행할 수 있습니다. 그리고 if와 else 사이에 else if의 약자인 elif라는 명령을 넣을 수 있는데요. 우리 책에서는 등장하지 않을 문법이지만 if, elif, else만 알고 있어도 재미있는 심리 테스트 게임을 만들 수 있습니다. 궁금한 사람은 인터넷에서 ‘파이썬 elif’라고 검색을 해보면 금방 이해할 수 있을 거예요!

```

print('[ 소름끼치도록 놀라운 심리테스트 ]')
menu = input('당신이 좋아하는 음식을 입력해주세요 : ')
if menu == '짜장면' :
    print('당신은 짜장면을 좋아하는 사람입니다.')
elif menu == '아이스크림' :
    print('당신은 아이스크림을 좋아하는 사람입니다.')
elif menu == '사탕' :
    print('당신은 사탕을 좋아하는 사람입니다.')
else :
    print('당신은 짜장면과 아이스크림과 사탕을 좋아하지 않는 사람입니다.')

```

## 5

### 순서 있는 저장 공간 리스트

드디어 파이썬 병아리반의 마지막 순서입니다. 우리 주변에 순서가 있는 데이터 종류로는 어떤 것들이 있을까요? 음식점에서 대기 명단에 이름을 적는 것에도 순서가 있고, 학번에도 순서가 있습니다.

파이썬에서는 순서가 있는 데이터를 다룰 때, 리스트(List) 데이터 구조를 사용합니다. 사실 우리는 리스트라는 것을 이미 앞에서 본 적이 있습니다. 바로 이 코드들에 있는 대괄호로 감싼 부분이 바로 리스트입니다.

```

for i in [0,1,2,3] :
    print(i ** 2)

```

```

names = ['초파', '루피', '상디', '조로']
for name in names :
    print(name)


```

여기에서 알 수 있듯이 리스트는 전체 데이터를 대괄호로 감싸주고 각각의 값들은 콤마로 구분합니다.


### ■ 리스트에 저장된 위치(index)로 값에 접근하기(indexing)

여러분은 `range()` 함수가 0부터  $n-1$ 까지의 연속된 정수를 생성했던 것을 기억할 겁니다. 프로그래밍을 하다 보면 1부터 숫자를 세는 것이 익숙한 사람과 달리 컴퓨터는 0부터 숫자를 세는 경우를 많이 볼 수 있는데요. 그러면 이 코드의 실행 결과는 무엇일까요?

```
names = ['초파', '루피', '상디', '조로']  
print(names[1])
```

 [1]이라는 코드가 무엇을 의미하는지는 잘 모르겠지만, 컴퓨터는 보통 0부터 센다고 했으니 맨 앞에 있는 '초파'가 0번일 것 같네요. 그러면 '루피'가 출력되나요?

아주 훌륭합니다. 그러면 대괄호 안의 숫자가 무엇을 의미하는지 이해했나요?

 음... 리스트에 저장된 데이터의 위치인 것 같아요!

정확합니다. 저장된 위치, 즉 인덱스(index)로 리스트의 값에 접근할 수 있다는 것이 리스트의 가장 큰 특징입니다. 다음 코드와 실행 결과를 보면 더 쉽게 알 수 있습니다.

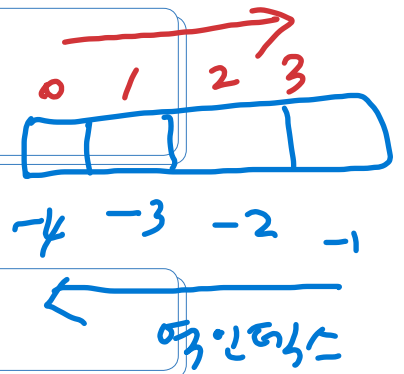
```
names = ['초파', '루피', '상디', '조로']  
print(names[0])  
print(names[1])  
print(names[2])  
print(names[3])
```

실행  
결과

초파  
루피  
상디  
조로

그리고 인덱스는 음수로 사용할 수도 있는데, 이때에는 ‘뒤에서 첫 번째’ 같은 개념으로 사용됩니다. 지금은 데이터 4개 밖에 없지만, 데이터가 많을 때는 앞에서 24번째를 세는 것보다 뒤에서 두 번째를 세는 것이 빠를 때가 있겠죠?

```
names = ['초파', '루피', '상디', '조로']  
print(names[-1]) # 인덱스 -1: 뒤에서 첫 번째 데이터에 접근
```



실행  
결과

조로

■ 리스트에 저장된 위치로 데이터의 일부 자르기(slicing)

리스트의 또 다른 특성은 순서에 따라 여러 데이터에 접근할 수 있다는 것입니다. 이것을 슬라이스(slice)라고 하는데요, 다음 코드처럼 사용할 수 있습니다.

```
names = ['초파', '루피', '상디', '조로']  
print(names[0:2])  
print(names[1:3])  
print(names[1:])  
print(names[:])
```



#### 실행 결과

```
['쵸파', '루피']  
['루피', '상디']  
['루피', '상디', '조로']  
['쵸파', '루피', '상디', '조로']
```

슬라이스를 할 때도 `range()` 함수와 마찬가지로 `[a:b]`는 `a` 이상 `b` 미만이라는 구간이 적용되는 것을 볼 수 있으며, 생각할 경우 ‘맨 앞에서부터’ 또는 ‘끝까지’와 같은 기능을 하는 것을 볼 수 있습니다.

#### ■ 리스트에 값 추가하기

이제 주어진 리스트에 값을 추가하는 방법을 알아보겠습니다. `append`라는 단어를 사전에서 찾아보면 ‘덧붙이다’, ‘첨부하다’라는 뜻입니다. 그래서 리스트에 값을 추가할 때는 다음과 같이 `append`라는 함수를 사용하면 됩니다.

```
names = ['쵸파', '루피', '상디', '조로']  
names.append('나미') # 리스트에 값 추가하기  
print(names)
```

#### 실행 결과

```
['쵸파', '루피', '상디', '조로', '나미']
```

그리고 리스트에 몇 개의 데이터가 저장되었는지 확인하려면 `length`라는 뜻의 `len()` 함수를 사용하면 됩니다. `len()` 함수는 꼭 리스트의 길이만을 알 수 있는 것은 아니고, 문자열의 길이도 알 수 있습니다.

실행  
결과

```
print(len(names))                # 리스트 길이 출력하기  
print(len('data analysis for everyone')) # 문자열 길이 출력하기
```

```
5  
26
```

지금까지 배운 내용들을 종합하면 이런 코드를 작성할 수 있습니다. 코드를 읽고 어떤 결과가 출력될지 예상해본 다음 직접 코드를 작성해서 결과를 확인하세요.

```
names = ['초파', '루피', '상디', '조로']  
names.append('해적왕')  
for name in names :  
    if len(name) > 2 :  
        print(name, '왔나요~?')
```

여기까지 파이썬 병아리반 내용이었습니다. 짧은 시간 동안 압축해서 배웠기 때문에 이해가 잘 안 되는 부분도 있을 수 있습니다. 그런 부분은 직접 다양한 값을 바꿔보며 연습하세요.

여기에서 다른 내용 외에 while 반복문, 딕셔너리 등도 알아둘 필요가 있습니다. 이 책에서 미처 다루지 않은 내용들은 파이썬 입문서 또는 인터넷 검색으로 학습할 것을 권장합니다.

파이썬 기초를 배웠으니 활용한 데이터 분석 프로젝트를 시작해 보세요!