Big Data Analytics Programming

Week-02. Python Programming Part - I

Jungwon Seo, 2020-Fall

'은/는'과 '이/가'의 차이를 설명 할 수 있나요?

버려진 섬에도 꽃은 핀다

- 김훈, 칼의 노래 中

배울 내용

Week-02. Python Basic

- Python 동작 방식
- 변수와 데이터 타입
- 조건문: if-statement
- 그룹형 데이터 타입
- 반복문: loop-statement

Levels of Programming Language

High Level? Low Level?

- 프로그래밍에서의 이해관계자 (Stakeholders)
  - 사람 (개발자)
  - 컴퓨터
- 작성은 우리가 하되, 컴퓨터가 알아 들을 수 있어야 함
  - 파이썬의 경우 interpreter가 이 역할을 함
  - hello =>1001010
  - 아나콘다 설치와 함께 파이썬 인터프리터가 설치됨

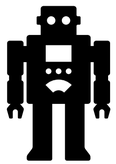
High

Python, Javascript, C, Java, C++ ....

- 인간(프로그래머) 친화적
- 인간(프로그래머)이 이해가 쉬움
- 디버깅(Debugging)이 쉬움
- 메모리 비효율성
- 실행을 시키기 위해서는, interpreter나 compiler가 필요

Assembly, Machine Language

- 기계(컴퓨터) 친화적
- 인간이 이해하기 어려움 = 컴퓨터가 이해하기 쉬움
- 디버깅이 어려움
- 메모리 효율성
- 어셈블리 언어에 대해서는 어셈블러가 필요

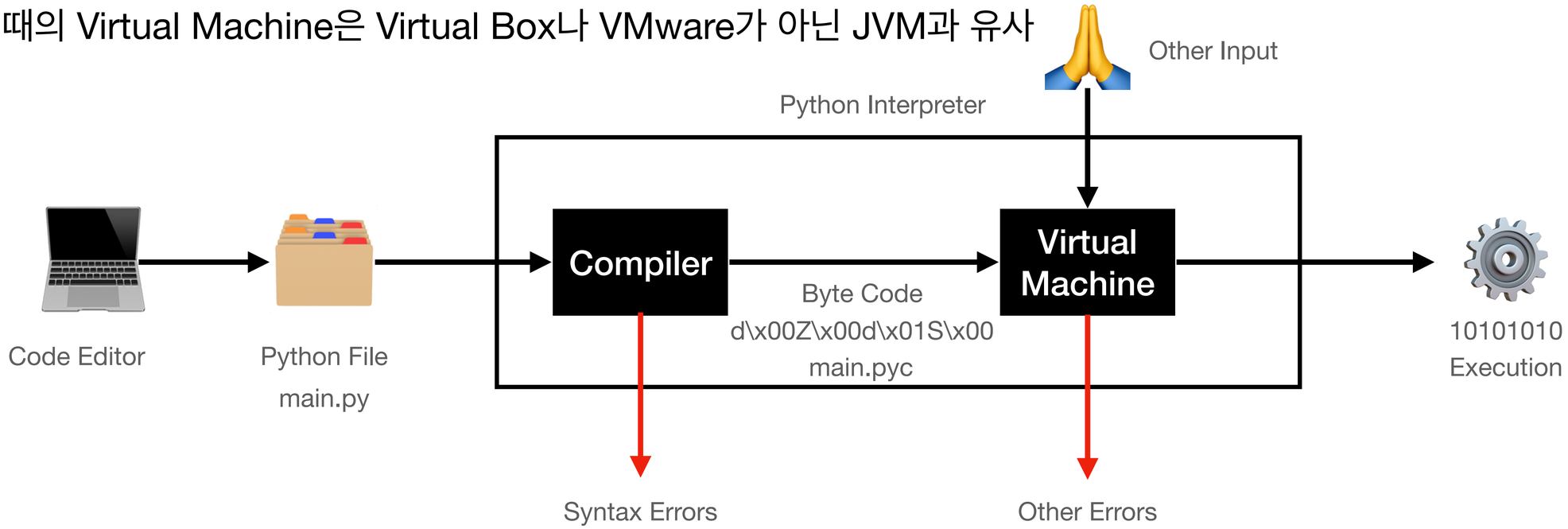


Low

How Python Works?

코드를 작성했다.. 그 다음은?

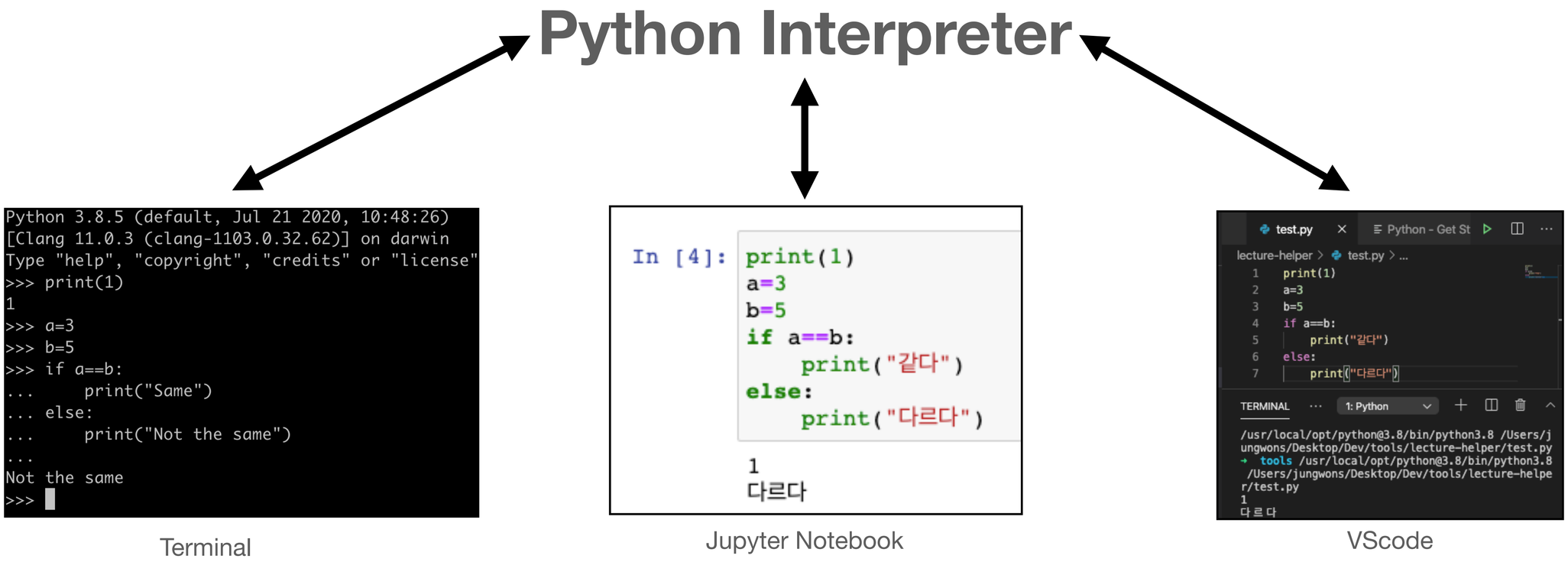
- 작성된 코드는, 파이썬 인터프리터를 통해서 Byte Code로 컴파일 된후 Virtual Machine에 의해 실행이 된다.
  - Virtual Machine에 의해 Python은 OS independent 하게 사용 가능하다.
  - 이때의 Virtual Machine은 Virtual Box나 VMware가 아닌 JVM과 유사



Python 코딩 환경

Editor는 결국 사람을 위한 것일 뿐..

- 어떤 환경에서 작업을 하든, 파이썬 코드는 인터프리터를 통해 동작을 한다.



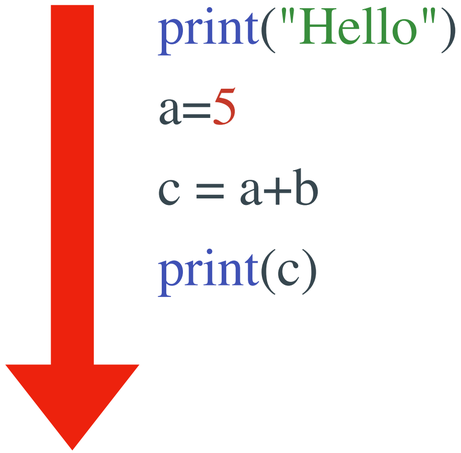
Terminal

Jupyter Notebook

VScode

코드 실행 과정
머리부터 발끝까지

- 기본적으로, 파이썬 코드는 위에서 아래 방향으로 코드가 실행된다.
  - * 내부적으로는 다른 언어와 마찬가지로 Stack 구조를 가져간다.
- 항상 명심해야 될 것은, 위쪽에서 정의(define) 되지 않은 내용을 아래쪽에서 다룰 수 없다.
  - 오른쪽의 코드는 실행 가능한가?
- “Rules… without them we live with the animals”
  - John Wick 中



```
print("Hello")
a=5
c = a+b
print(c)
```

Python Errors - Part 1
에러의 종류

- Syntax Error
  - 파이썬 코드가 Byte code로 변환되는 과정중에 감지됨
  - 다르게 말해, “문법에 맞지 않게 썼다.” 의 의미

```
a 5
```

```
  File "<ipython-input-6-1a22f7fb4338>", line 1
    a 5
      ^
SyntaxError: invalid syntax
```

- Name Error
  - 해당 변수가 발견되지 않았을 경우
  - “처음 보는 변수인데?” 의미

```
print(k)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-eb2fa875d160> in <module>
----> 1 print(k)

NameError: name 'k' is not defined
```

에러에 관한 자세한 설명은
https://docs.python.org/3/library/exceptions.html

Python Errors - Part 1
에러의 종류

- Type Error
  - 어떠한 연산을 할 때, 연산이되어지는 변수들의 변수형(Type)이 호환되지 않았을 경우
  - 1+"가"=?

```
1+"a"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-4ddca89a012c> in <module>
----> 1 1+"a"

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

- Indentation Error
  - 파이썬의 경우에는 변수/수행의 범위를 tab을 기준으로 나눔
  - 들여쓰기를 잘못 사용

```
if 5==5:
print(1)
```

```
  File "<ipython-input-9-269a27d2924a>", line 2
    print(1)
        ^
IndentationError: expected an indented block
```

언어를 배울 때 가장 먼저 배우는 것은?

ㄱㄴㄷㄹㅁㅂㅅㅇㅈㅊㅋㅌㅍㅎ

ㄲㄸㅃㅆㅉ

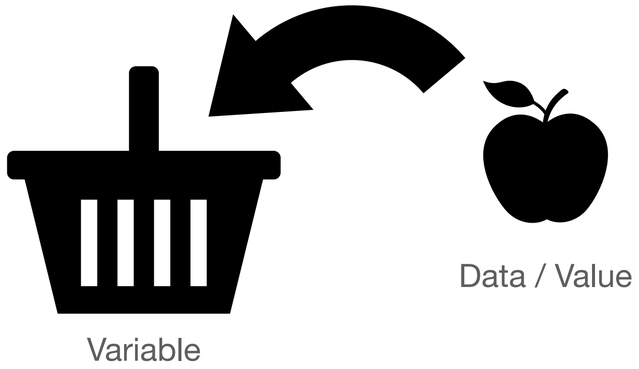
ㅏㅑㅓㅕㅗㅛㅜㅠㅡㅣ

ㅐㅒㅔㅖ

ㅘㅙㅝㅞㅢ

Python 변수
Variable과 Data Type

- 변수 (variable): 메모리에 값을 저장(=)하기 위한 명명(Named)된 위치
  - **Equal sign(=)은 같다의 의미가 X**
  - <변수명> = <값>
  - a = 1
  - my_name = "Jungwon"
- 동일 한 변수명에 여러번 값을 대입할시,
  - 최종 대입 값으로 대체된다.
- 변수명 작성시 주의할점
  - 숫자로 시작 X, 띄어쓰기 X
  - 한글도 가능, 하지만 아무도 안씀



Variable = data

```
a = 10
a = "Hello"
a = True
print(a)
```

동적 타이핑 vs 정적 타이핑

Python은 동적 타이핑(Dynamic Typing)을 사용하므로, 런타임시 변수의 타입을 결정

```python
sum = 0
for i in range(0, 10):
    sum += i
```

Python

```javascript
var sum=0;
for (var i = 0; i < 10; i++) {
  sum += i
}
```

Javascript

```cpp
int sum = 0;
for (int i = 0; i < 10; i++){
    sum += i;
}
```

C++, Java

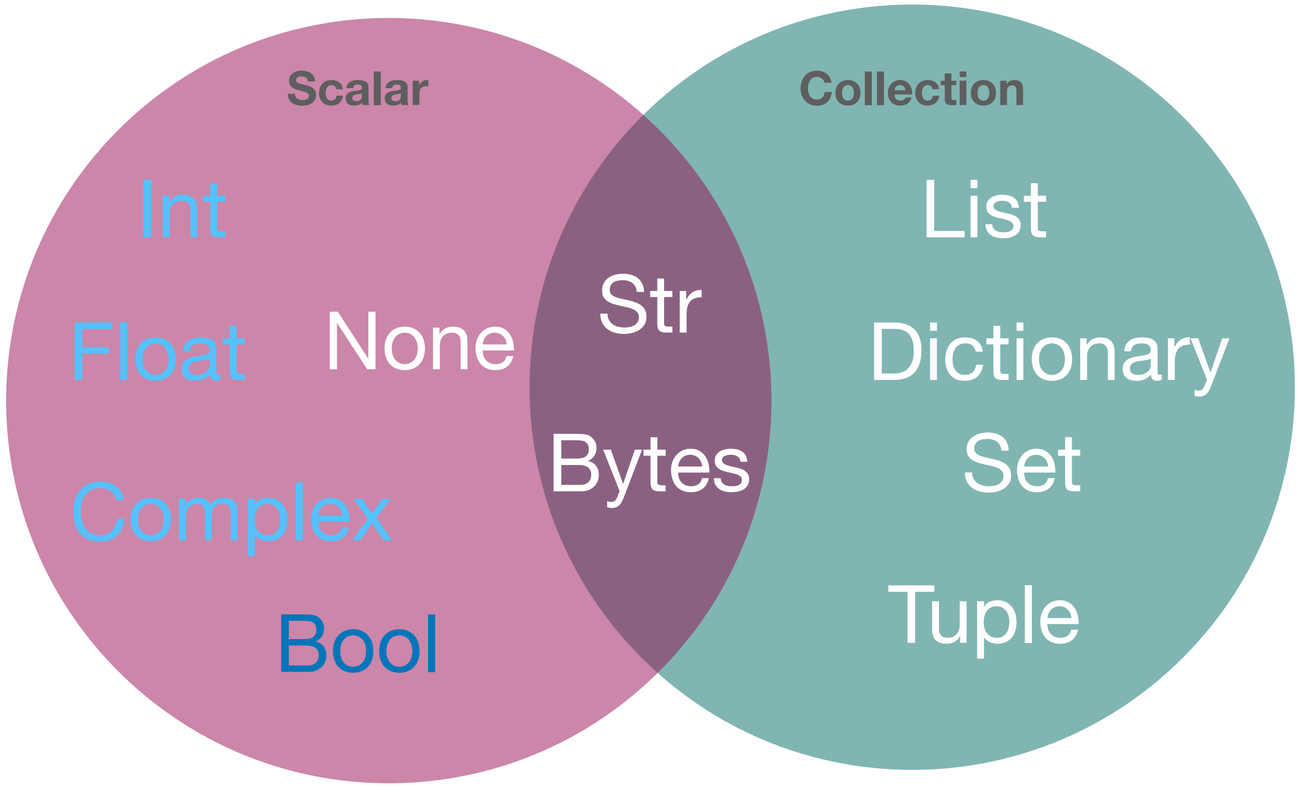
```go
var sum int = 0
for i := 0; i < 10; i++ {
    sum += i
}
```

Go

Python Built-in Data Type

Scalar and Collection

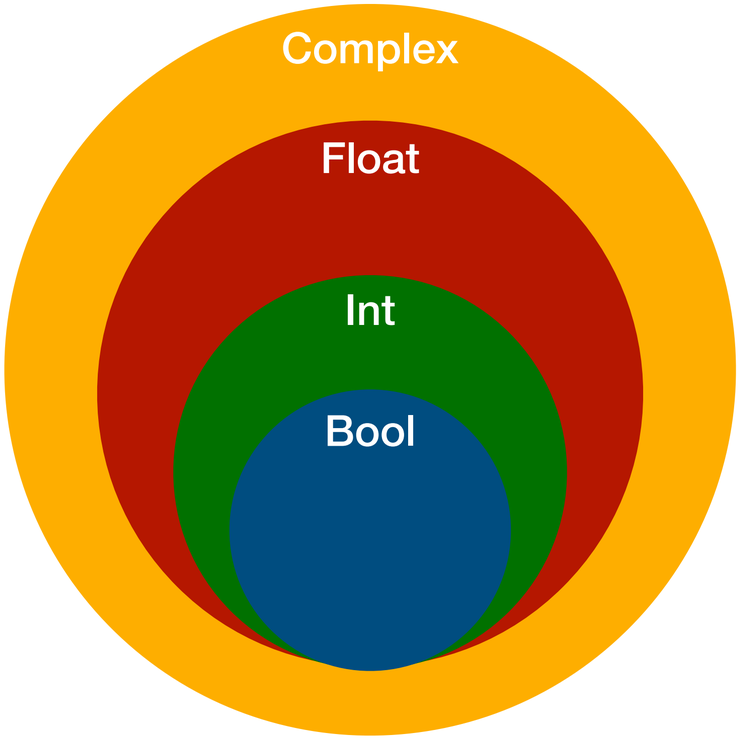
- Scalar: 단일 값을 갖음
- Collection: 값"들"을 갖음



Python 변수에 관한 자세한 설명
https://docs.python.org/3.7/library/stdtypes.html#

Python Data Type - Scalar
숫자형 Data Type (Numerical)

- 세부 종류: int, float, complex , (bool)
- 사칙연산의 경우 일반 수학적 연산과 동일하게 동작
  - 3 + 3 = 6, 3 - 3 = 0, 3 * 3 = 9, 3 / 3 = 1.0
  - 추가적인 연산자들
    - %: 모듈러 연산자로 나머지 값(Remainder)을 출력 => **5%3 = 2**
    - //: 내림(floor)가 포함된 나누기 연산 = 몫(quotient)을 출력 => **5//3 = 1**
    - **: 지수(exponential) 연산 => **5**2 = 25**
    - &,| : 비트연산 => 이진수로 변환뒤 연산
- 부울형 (boolean) : True or False
  - Bool 변수라고 불리우며, 우리가 흔히 말하는 1과 0
  - 사칙 연산과 논리 연산 제공, 단 사칙연산 적용시 int나 float으로 자동형변환
- 숫자형 변수간의 연산시 형변환
  - 연산변수간의 타입의 다를 경우 또는 연산결과의 타입이 다를 경우 상위 변수형으로 형변환



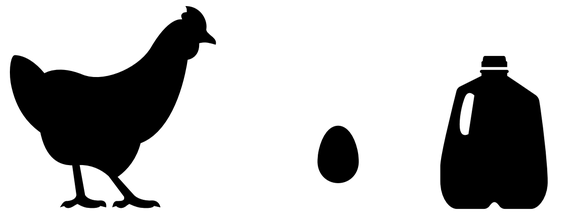
Python Data Type - Scalar
문자형 Data Type (String)

- 큰 따옴표(") 또는 작은 따옴표(')로 값을 감싸줘서 표현
  - 1: int, "1":string
  - "abcde", 'abcde'
- 문자형: 더하기(Addition) 연산만 지원
  - 이때의 더하기 연산은 이어쓰기(concatenate)의 효과
  - "hello" + "world" = "helloworld"
- Byte string

논리적으로 생각한다는 것은?

부모님께서 프로그래머인 나에게 심부름을 시키셨다.
"슈퍼가서 우유 하나 사와. 아, 계란 있으면 6개사와"
그래서 우유를 6개 사갔다.
"왜 우유를 6개나 샀어?"
"계란이 있길래.."

- From 인터넷 어딘가..



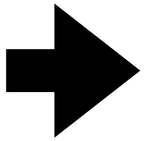
조건문(conditional statement)

What **if…** anything **else?**

- 사람은 항상 특정한 조건/상황에 기반해서 판단
- 세상에서 일어나는 모든 상황을 Logically 코드로 표현 가능

If I were a boy
Even just for a day
I'd roll outta bed in the morning
and throw on what I wanted and go

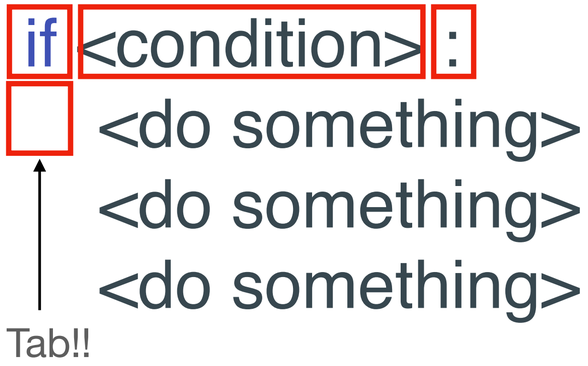
- *Beyonce, If I Were A Boy* 中



```
if user.gender == "boy" and user.duration >= "1":
    user.wake_up(from="bed", when="morning")
    user.wear(style="whatever")
    user.go_out()
```

조건문(conditional statement)
If문 사용법

- 조건문 작성법
  - if 라는 예약어로 시작
  - If 뒤에는 True or False로 나올 수 있는 조건
  - 해당 하는 조건의 ending 은 콜론 (:)으로 마무리
  - 그 조건문이 "만족" 됐을 때 동작할 코드는 들여쓰기(indentation) 이후 작성
  - If 안에서만 동작해야할 코드는 들여쓰기를 유지



조건문(conditional statement)

True, False?

- 기본적인 Boolean 타입은 그대로 유지 : True is True, False is False
- Boolean 타입 외에도 조건으로 사용가능

Team True!! 👻

True

All numbers except 0

All collections except
collection **with no element**

~~All functions~~

Team False!! 👽

False 0 0.0

None [ ]

( )

" " { }

조건문(conditional statement)

비교에 의한 조건

- 값들간의 비교를 통해서 조건을 만들 수 있다.
  - A가 B보다 크다 : **A > B**
    - A=5, B=3 일때의 결과는?
    - A=3, B=5 일때의 결과는?
  - A가 B보다 작다 : **A < B**
  - A가 B보다 크거나 같다: **A >=B**
  - A가 B보다 작거나 같다: **A <=B**
  - A와 B가 같다: **A == B, A is B**
  - A와 B가 다르다: **A != B, A is not B**
- 결과는 결국 True or False로 반환

```
A=5
B=3
print(A>B)
print(A<B)
print(A>=B)
print(A<=B)
print(A==B)
print(A is B)
print(A is not B)
```

조건문(conditional statement)
조건들의 조합

- 단순 1차원적인 조건이 아닌, 여러 조건을 연결 가능
- A는 B보다 크면서 0이 아닐다
  - A>B and A!=0
- A는 B보다 크거나 100이다
  - A>B or A==0
- 다른 언어에서는 &&, 난 ||로 and, or 가 표현됨
  - &와 |는 bit연산으로 사용됨
  - 결과적으로 각각의 조건이 True or False로 나오는 상황에서는 and, or와 동일하게 동작

```
A=5
B=3
print(A>B and A==0)
print(A>B and A!=0)
print(A>B or A==0)
print(A>B or A!=0)
print(A>B & A==0)
print(A>B | A==0)
```

조건문(conditional statement)

조건은 Optional 일 수 있고, Alternative 있다

- 단순히 특정 케이스에 대해서만 반응을 하기 위해서는
  - If와 elif (else if)로 조건문을 사용 할 수 있다.
  - 예) 만약 점수가 100점 이상이면 100점이라고 친다
  - 예) 만약 점수가 100점 이상이면 100점이라고 치되,
    0점 이하면 0점으로 친다.

```
score = 105
if score >= 100:
    score = 100
elif score <= 0 :
    score = 0
```

- 특정 케이스가 아닌 케이스에 대해서도 반응을 하기 위해서는
  - else로 전체 조건문을 마무리 할 수 있다.
  - 예) 만약 점수가 50점 이상이면 Pass라고 한다, **그게 아니라면** Fail이다.

```
score = 70
grade = None
if score >= 50:
    grade = "Pass"
else:
    grade = "Fail"
```

영어를 배웠을 때 복수형을 배웠듯이..

s나 es를 붙인다.
y를 i로 바꾸고 es를 붙인다.
f나 fe를 v로 바꾸고 es를 붙인다
……

Python Data Type - Collection
묶음형 Data Type

- 단일 값만 저장 하는 것이 아니라, 값들을 저장
  - 복수 값 저장의 나쁜 예: val1 = 1, val2 = 2, val3 = 3 …..
  - 복수 값 저장의 좋은 예: my_list = [1, 2, 3]
- 목적에 따라서 다양한 묶음 형 데이터 타입을 사용
  - List: [1,2,3,4,5,6]
  - Dictionary: {"name":"Jungwon", "score":100}
  - Tuple: (1,"b")
  - Set : {1,2,3}
- 값들의 묶음이기 때문에, string은 list와 유사한 특징을 띔
  - my_list = ['a','b','c']
  - my_str = 'abc'

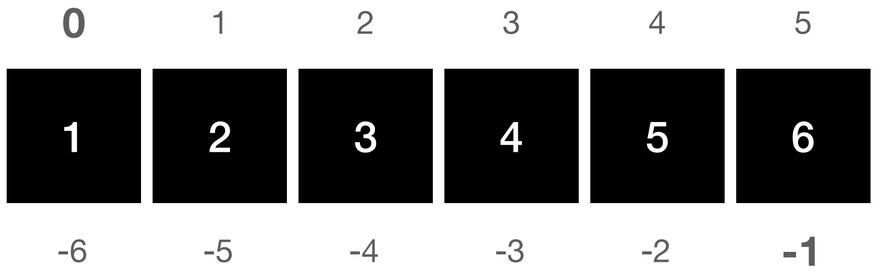
Python Data Type - List
array? 배열? 인덱스?

- list를 표현 하는 기호는 대괄호(Brackets): [ ]
- 원소들을 구분하는 기호는 쉼표(comma): ,
  - [ ] : 원소가 없다
  - [1]: 원소가 1개
  - [1, 2]: 원소가 2개
  - [0]: 원소가 몇개?
- 원소는 object 또는 function이라면 가능
  - 단일 값: [1,2,3], ['a','abc','d']
  - 묶음 형: [[1,2,3], [1,2,3]] , [{"name":"손흥민"}, {"name":"류현진"}]
  - 함수 : [function1, function2, function3]

Python Data Type - List
CRUD - Create, Read, Update, Delete

- Create
  - **my_list = [ ]** 또는 **my_list = list( )**, 만약 초기값(initial value)를 제공한다면, **my_list = [1,2,3]**
- Read
  - list 전체에 대한 접근은 list 명 그대로: **print(my_list), temp_list = my_list**
  - 원소에 대한 접근은: **my_list[*index*]**, 이때 index는 해당 원소의 위치 (0부터 시작, 정수)
  - 예) **a = [1, 2, 3, 4]** 일 때 **a[3] = ?**
- Update
  - list 전체를 변경하는 경우는 변수 재할당 : **my_list = ['a','b','c']**
  - 원소에 대한 변경은: **my_list[*index*] = 'k'**
  - 원소 추가는: **my_list.append('x')**
  - Concatenation은 : **[1,2,3] + ['a','b','c'] = [1, 2, 3, 'a', 'b', 'c']**
- Delete
  - list 전체에 대한 delete는: **del my_list**
  - n번째 원소에 대한 delete는 : **del my_list[*n*]**

Python Data Type - List
Slicing



- Python의 List의 경우에는 Subset에 접근 가능
  - 예) **a = [1,2,3,4,5,6]** 일때 앞에 3개만 추출 하려는 경우?
- list[start:end]
  - start는 시작하려는 index, end는 끝나기 전의 index
  - **a = [1,2,3,4,5,6]** 일때, **a[0:3]**은 0,1,2 index에 대한 접근 => **[1,2,3]**
  - 만약 start 부분 또는 end 부분을 생략한다면, **list[:] == list [0 : len(list)]**
- list[-start:-end]
  - 음수의 index를 넣는 경우, "뒤에서부터 n번째"의 의미
  - 예) a[-1]는 맨 뒤의 원소, a[-3]은 뒤에서 세번째 원소
  - 예) a[1:-3]은?

Python Data Type - Dictionary
사전? Key-value? map?

- dictionary를 표현 하는 기호는 중괄호(brace): { }
- List가 index 기반으로 데이터를 Mapping 시킨다면, Dictionary는 key 기반으로 데이터를 Mapping 시킴
  - { key: value }
  - player = { "name":"손흥민", "team":"토트넘", "height":183 }
  - my_dict = { "a" : 3, 1: "c", (1,2,3): 4}
- Key는 중복될 수 없지만, Value는 중복될 수 있음
  - { "a": 123, "a": 456 } => 😰
  - {"a":123, "b":123} => 😀
- Key와 value는 object 또는 function이라면 가능
  - 단 **mutable** 한 list, dictionary, set은 Key로 불가능

Python Data Type - Dictionary
CRUD - Create, Read, Update, Delete

- Create
  - **my_dict = { }** 또는 **my_dict = dict( )**, 만약 초기값(initial key-value)를 제공한다면, **my_dict = { "a":1 }**
- Read
  - dictionary 전체에 대한 접근은 dictionary 명 그대로: **print(my_dict), temp_dict = my_dict**
  - Value에 대한 접근은: **my_dict[*key*]**, 이때 key는 dictionary를 정의 할 때 사용한 Key
- Update
  - dictionary 전체를 변경하는 경우는 변수 재할당 : **my_dict = { "a": 3 }**
  - Value에 대한 변경은: **my_dict[*key*] = *value***
  - Key-Value 추가: **my_dict[*new_key*] = *value***
- Delete
  - dictionary 전체에 대한 delete는: **del my_dict**
  - Key에 대한 delete는 : **del my_list[*key*]**
    - 이때, Mapping 된 Value도 같이 소멸

Python Data Type - Dictionary

Keys, Values, Items

- Dictionary의 전체 데이터에 대한 접근은 크게 세가지
- 전체 Key 출력
  - my_dict.keys( )
  - 출력: dict_keys(['name', 'age', 'height', 'retired'])
- 전체 Value 출력
  - my_dict.values( )
  - 출력: dict_values(['손흥민', 29, 183, False])
- 전체 Item 출력
  - my_dict.items( )
  - 출력: dict_items([('name', '손흥민'), ('age', 29), ('height', 183), ('retired', False)])

```
{
    'age': 29,
    'height': 183,
    'name': '손흥민',
    'retired': False
}
```

Python Errors - Part 2
에러의 종류

- Index Error
  - list와 같은 index기반의 접근을 하는 데이터에 접근 가능한 범위 밖의 값을 접근하려는 경우
  - a의 원소가 3개인데 100번째 원소는?

```
a = ["a","b","c"]
a[100]
```

```
---------------------------------------------
IndexError                                ]
<ipython-input-1-268446cb9d24> in <module>
      1 a = ["a","b","c"]
----> 2 a[100]

IndexError: list index out of range
```

- Key Error
  - Dictionary와 같이 Key기반으로 접근하는 데이터에 존재 하지 않는 Key로 접근하려는 경우
  - key가 a,b,c 밖에 없을 때 d에 대한 값은?

```
a = {"a":1, "b":2, "c":3}
a["d"]
```

```
---------------------------------------------
KeyError                                  ]
<ipython-input-2-7f438077e339> in <module>
      1 a = {"a":1, "b":2, "c":3}
----> 2 a["d"]

KeyError: 'd'
```

Python Data Type - Tuple
Ordered, Immtuable

- Tuple을 표현하는 기호는 소괄호(Parentheses) : ( )
  - my_tuple = (1,2,3)
  - Index 기반으로 접근 가능
- List와 비슷한 형태를 띄지만 몇가지 특징이 있음
  - Immutable 데이터 타입으로, 값의 변경이 불가
  - Immutable 데이터 타입이므로 Dictionary의 Key로 사용 가능
  - 함수에서 여러개의 값을 return 하는 경우에 사용됨
    - * list, dict등은 여러 원소가 있는 것이지 return 되는 값은 1개

```
def hello():
    return 1, 2

print(type(hello()))
```

```
<class 'tuple'>
```

튜플에 관한 자세한 설명은
https://openbookproject.net/thinkcs/python/english3e/tuples.html#tuples

Python Data Type - Set
수학의 집합과 유사, Unordered and Unindexed

- Set을 표현하는 기호는 중괄호(brace) : { }
  - Dictionary와 같은 기호를 사용하지만, 원소가 key-value 형태로 저장되느냐에 의해 구별됨
- 몇가지 특징
  - Unordered: 순서가 보장되지 않음
  - Unindexed: index 기반으로 값이 접근 가능하지 않음
  - Unique Element: 원소가 중복되지 않음
- 자주쓰이는 사용처
  - List에서 Unique값 추출 시 (중복제거시)

```
my_list = [3,1,33,1,5,55]
my_set = set(my_list)
print(my_set)
```

```
{1, 33, 3, 5, 55}
```

Set의 관한 자세한 사용법은
https://www.w3schools.com/python/python_sets.asp

반복문 (iterative statement)

While I am… **For** how long?

- 특정 조건을 만족하는 한, 그 구간을 반복한다.
  - **While** 문을 활용
  - 예1) budget이 0이상인 동안
  - 예2) 에러가 3번 이하로 나는 동안

```
budget = 100
while budget > 0 :
    budget -= buy()
print("No more money")
```

Tab!!

```
num_of_errors = 0
my_file = None
while num_of_errors < 3:
    try:
        my_file = download_file()
    except:
        num_of_errors+=1
```

- Collection 안에 있는 원소들을 각각 접근한다.
  - **For** 문을 활용
  - range를 활용한 0 부터 n까지 접근
  - List와 같은 collection 데이터의 각각의 원소에 접근

```
for i in range(0,10):
    print(i)
```

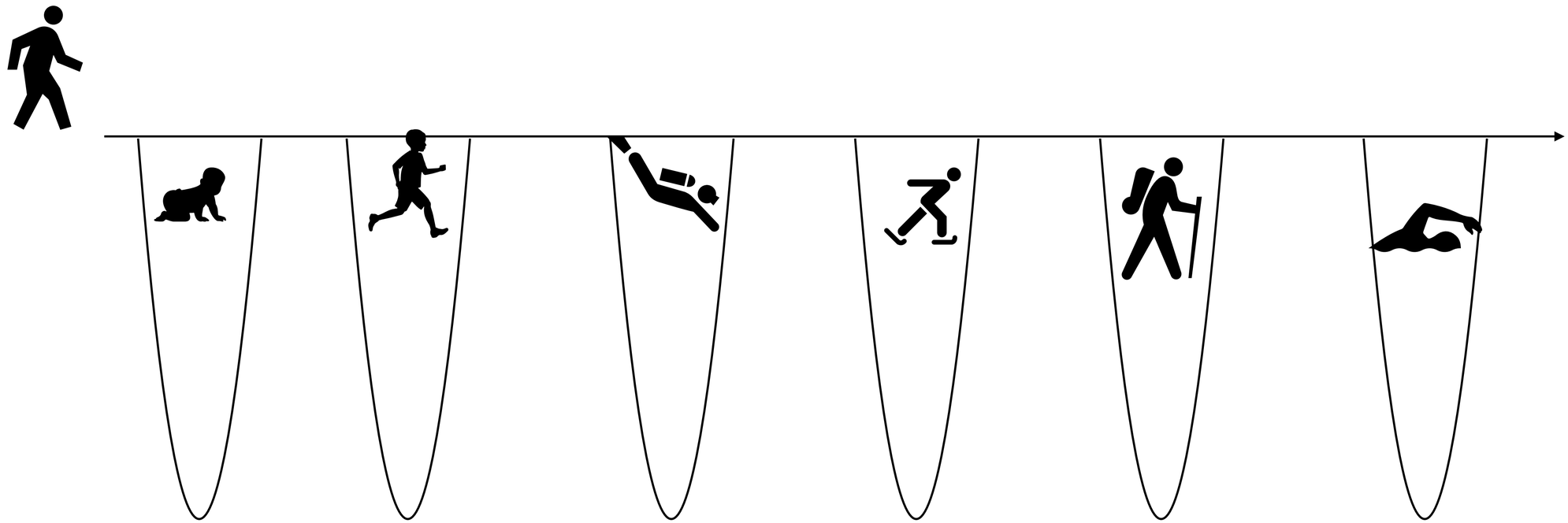
Tab!! Collection

```
my_list = ['a','b','c']
for ele in my_list:
    print(ele)
```

- while문 for문에 해당하는 영역은 마찬가지로 tab을 기준으로 구별

프로그래밍의 여정

골짜기(Valley)에 매번 빠질 것입니다...



E.O.D