AI 이노베이션스퀘어

-파이썬 프로그래밍-

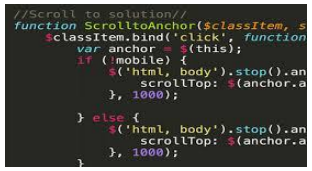
junheon@hanyang.ac.kr
전준헌

컴퓨터

- 과거의 컴퓨터
  - Compute: 계산하다
  - Computer: 계산하는 사람 또는 기계

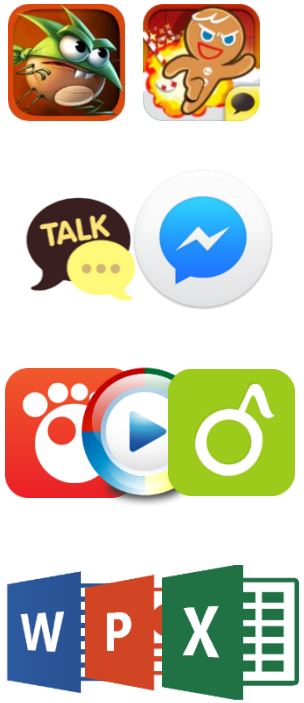
NASA 최초의 우주 궤도 비행 프로젝트

- 현재의 컴퓨터
  - 하드웨어 + 소프트웨어



0/1 기반
프로그램/데이터
(Software)

전자 기기
(Hardware)



⋮

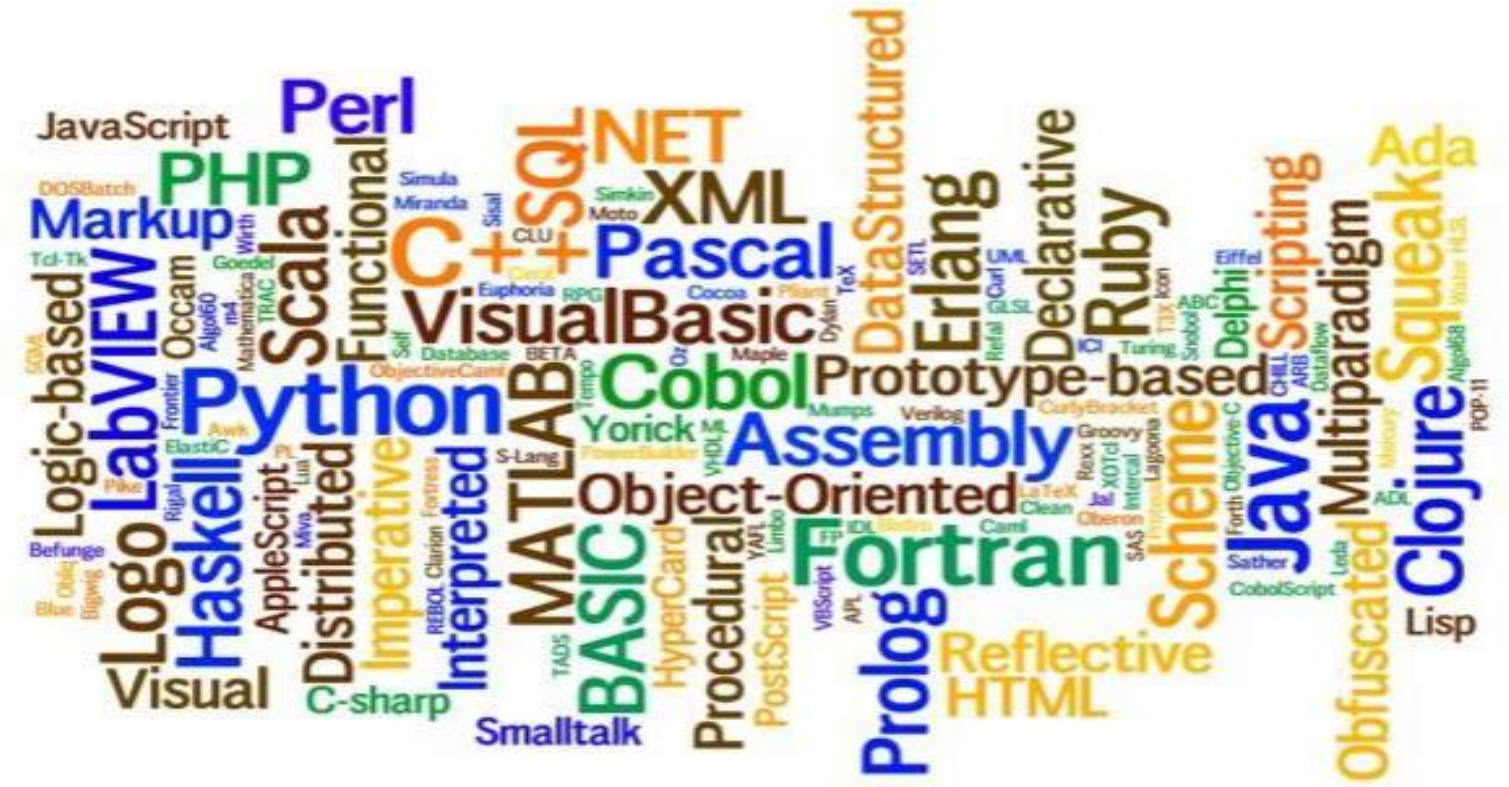
다양한 서비스

소프트웨어와 프로그래밍 언어

- 소프트웨어란?
  - 컴퓨터 하드웨어에게 필요한 기능을 수행할 수 있도록 하는 프로그램
- 프로그램이란?
  - 특정 목적을 위해 프로그래밍을 통해 만든 명령어들의 모임
- 프로그래밍이란?
  - 수식이나 작업을 컴퓨터가 수행하기 알맞게 정리해 순서를 정하고 컴퓨터 명령 코드인 프로그래밍 언어를 이용하여 코딩하는 작업을 총칭하여 쓰는 말
  - 코딩(Coding)이란?
    - 프로그래밍 언어를 사용하여 프로그램의 코드를 작성하는 일
    - 알고리즘을 프로그래밍 언어로 변환하는 작업

- 프로그래밍 언어란?
  - 사람과 컴퓨터 사이에 존재하는 일종의 **커뮤니케이션 수단**.

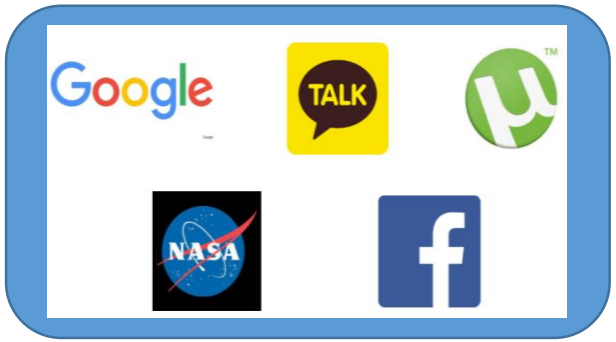
- 프로그래밍 언어마다 특징도 다양하고 장단점도 다양하다.



다양한 프로그래밍 언어

Why python?

왜 파이썬을 쓰는 거지?



파이썬을 사용하는 기업들

https://docs.python.org/ko/3/tutorial/index.html
https://docs.python.org/ko/3/tutorial/appetite.html

- 배우기 쉽다.
- 간결한 문법과 풍부한 표준 라이브러리를 가지고 있다.
- 자료 구조들과 객체 지향 프로그래밍에 대해 간단하고도 효과적인 접근법을 제공
- 인터프리터적인 특징들은 대부분 플랫폼과 다양한 문제 영역에서 스크립트 작성과 빠른 응용 프로그램 개발에 이상적인 환경을 제공
- 풍부한 표준 라이브러리는 소스나 바이너리 형태로 파이썬 웹 사이트, https://www.python.org/, 에서 무료로 제공, 자유롭게 배포 가능
- C 나 C++ (또는 C에서 호출 가능한 다른 언어들)로 구현된 함수나 자료 구조를 쉽게 추가할 수 있음

Why python?

- 웹, 앱에서 인공지능까지 인기가 급상승하며 주목받는 언어
- 데이터 분석에 장점을 가진 스크립트 언어
  - '데이터 사이언티스트'라는 직종이 인기가 있음
- 해외에서는 웹 어플리케이션의 개발 언어로도 많이 사용되고 있으며, Python 프로그래머의 평균 연봉이 높은 것이 화제가 되기도 한다.
- 최근에는 기계학습 등 인공지능의 개발에도 많이 사용되고 있다. 버전 2.x와 3.x는 일부 호환이 되지 않지만 두 버전 모두 이용자가 많다.

From 프로그래밍 언어 도감

Python의 특징

- 인터프리터
  - 프로그래밍 언어로 작성된 소스코드를 바로 실행할 수 있는 프로그램 또는 환경
  - 컴파일러(원시코드를 기계어로 해석해주는 프로그램 또는 환경)언어보다 속도가 느린 편
- 생산성
  - 간결한 문법과 풍부한 라이브러리 제공 등으로 개발하는데 있어 생산성이 뛰어남
- 멀티패러다임지원
  - 함수형 패러다임과 객체지향 패러다임 두 가지 모두를 지원(다양한 방식, 다양한 관점으로 프로그래밍할 수 있으며, 수많은 기법과 디자인 패턴을 접목시킬 수 있음)
- Glue Language
  - 다른 언어와의 호환성이 높음, C언어로 구현한 CPython, Java로 구현한 Jython, 파이썬으로 파이썬을 구현한 PyPy 등 대체 가능한, 특정 플랫폼에 맞춰진 구현체를 가지고 있음
- 다양한 라이브러리 제공
  - 수많은 표준 라이브러리를 기본으로 탑재, pip를 통해 손쉽게 받을 수 있음

Python 설치

- 아나콘다 설치
  - www.anaconda.com

- Python 설치
  - www.python.org

■ pip

○모듈이나 패키지를 쉽게 설치할 수 있도록 도와주는 도구 (De Facto)

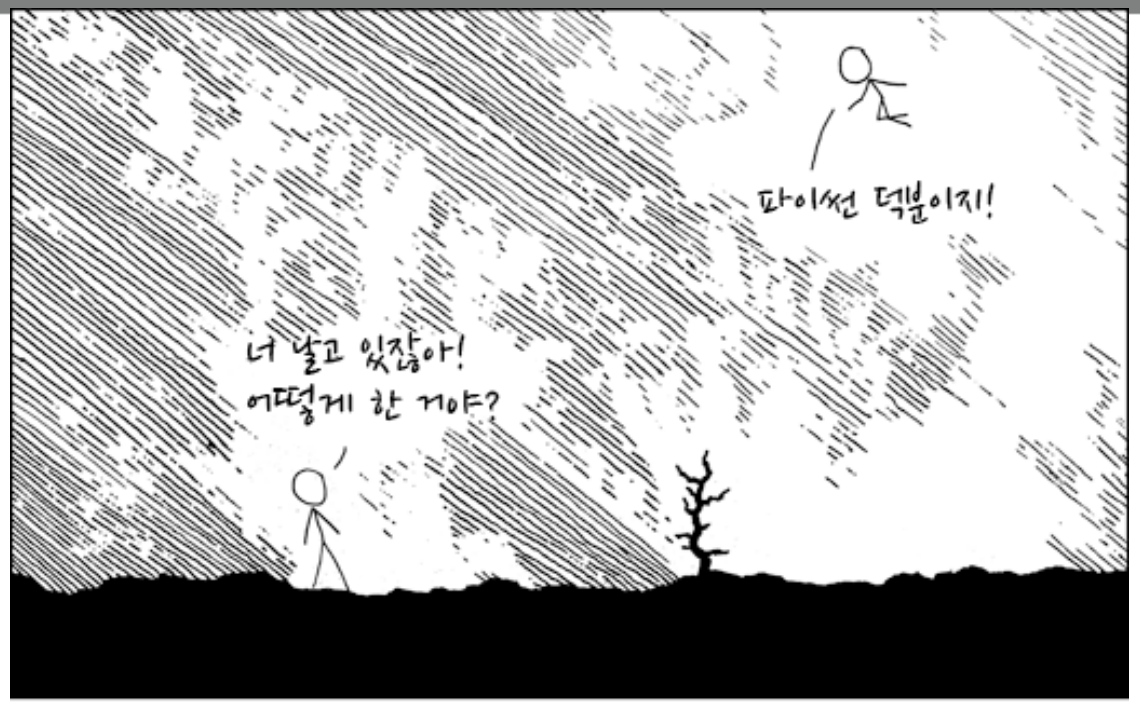
- ➢설치된 패키지 확인 pip list / pip freeze
  pip show '패키지 이름'
- ➢패키지 검색
  pip search '패키지 이름'
- ➢패키지 설치
  pip install '패키지 이름'
- ➢패키지 업그레이드
  pip intall –upgrade '패키지 이름' pip install –U '패키지 이름'
- ➢패키지 삭제
  pip uninstall '패키지 이름'

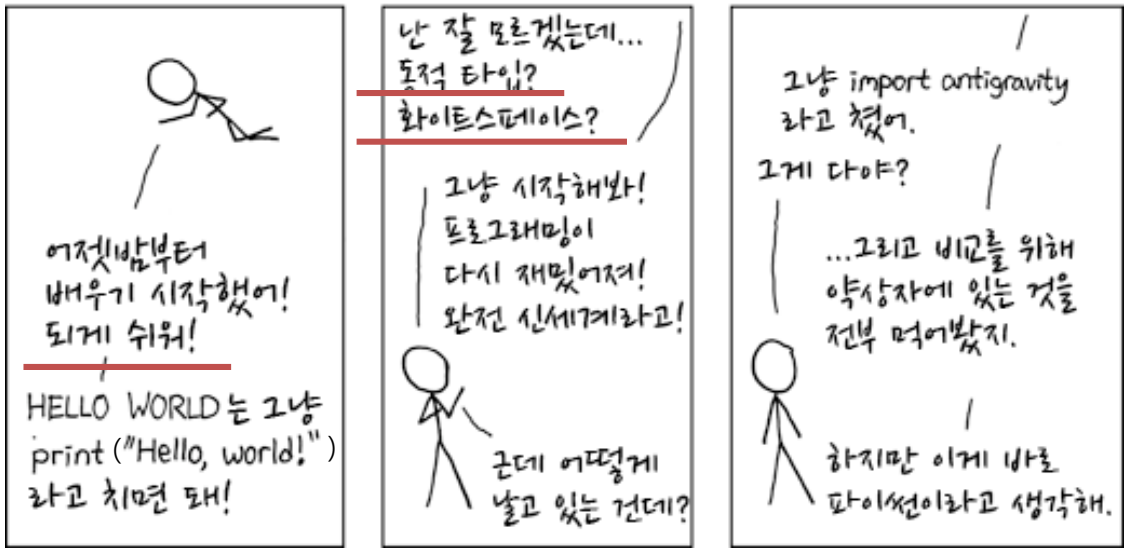
■ Anaconda

○데이터 분석에 필요한 패키지를 모아서 한번에 설치

Python의 General Purpose

In [ ] : import antigravity





C.f)
Domain-specific

In [ ]:
```
import this
```

■ 아름다운 것이 보기 싫은 것보다 좋다.
■ 명시적인 것이 암묵적인 것보다 좋다.
■ 간단한 것이 복합적인 것보다 좋다.
■ 복합적인 것이 복잡한 것보다 좋다.
■ 수평한 것이 중첩된 것보다 좋다.
■ 희소한 것이 밀집된 것보다 좋다.
■ 가독성이 중요하다.
■ 규칙을 무시할 만큼 특별한 경우는 없다.
■ 하지만 실용성이 순수함보다 우선한다.
■ 에러가 조용히 넘어가서는 안된다.
■ 명시적으로 조용히 만든 경우는 제외한다.
■ 모호함을 만났을 때 추측의 유혹을 거부해라.
■ 하나의 — 가급적 딱 하나의 — 확실한 방법이 있어야 한다.
■ 하지만 네덜란드 사람(귀도)이 아니라면 처음에는 그 방법이 명확하게 보이지 않을 수 있다.
■ 지금 하는 것이 안하는 것보다 좋다.
■ 하지만 안하는 것이 이따금 지금 당장 하는 것보다 좋을 때가 있다.
■ 설명하기 어려운 구현이라면 좋은 아이디어가 아니다.
■ 설명하기 쉬운 구현이라면 좋은 아이디어다.
■ 네임스페이스는 아주 좋으므로 더 많이 사용하자!

프로그래밍이란?

문법(키워드, 식,문)을 이용해서
값을 입력받고, 계산/변환하고, 출력하는 흐름을 만드는 일

키워드(keyword)?
표현식(expression)?
문(statement)?
값(value)?

```
import keyword
keyword.kwlist
```

- 예약어(reserved word, keyword)

False	elif	lambda
None	else	nonlocal
True	except	not
and	finally	or
as	for	pass
assert	from	raise
break	global	return
class	if	try
continue	import	while
def	in	with
del	is	yield

Expression

■ 표현식(expression) = 평가식 = 식

○값

○값들과 연산자를 함께 사용해서 표현한 것

○"평가"되면서 하나의 특정한 결과값으로 축약

➢ 수

– 1 + 1

» 1 + 1 이라는 표현식은 평가될 때 2라는 값으로 계산되어 축약

– 0 과 같이 값 리터럴로 값을 표현해놓은 것

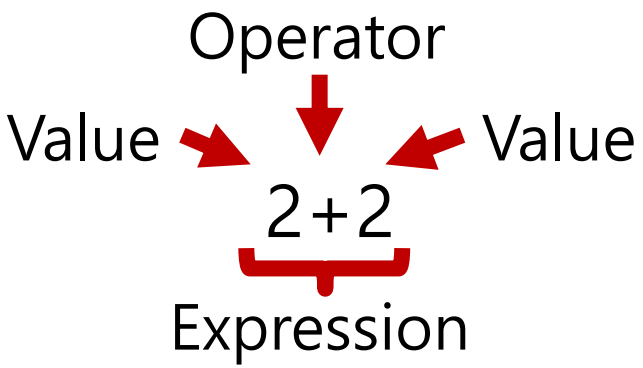
➢ 문자열

– 'hello world'

– "hello" + ", world"

○궁극적으로 "평가"되며, 평가된다는 것은 결국 하나의 값으로 수렴한다는 의미

➢ python에서는 기본적으로 left-to-right로 평가



2+2	Addition
2-2	Subtraction
2*2	Multiplication
2/2	Division
2%2	Modulus
2**2	Square

- +, -, *, / and % are called **operators.**
- * sign is called an **asterisk.**

```
In [ ] : 2+2
Out[ ] :
```

```
In [ ] : 5-7
Out[ ] :
```

```
In [ ] : 8*9
Out[ ] :
```

```
In [ ] : 10/2
Out[ ] :
```

```
In [ ] : 3**2
Out[ ] :
```

```
In [ ] : 10%2
Out[ ] :
```

```
In [ ] : 2  +  2
Out[ ] :
```

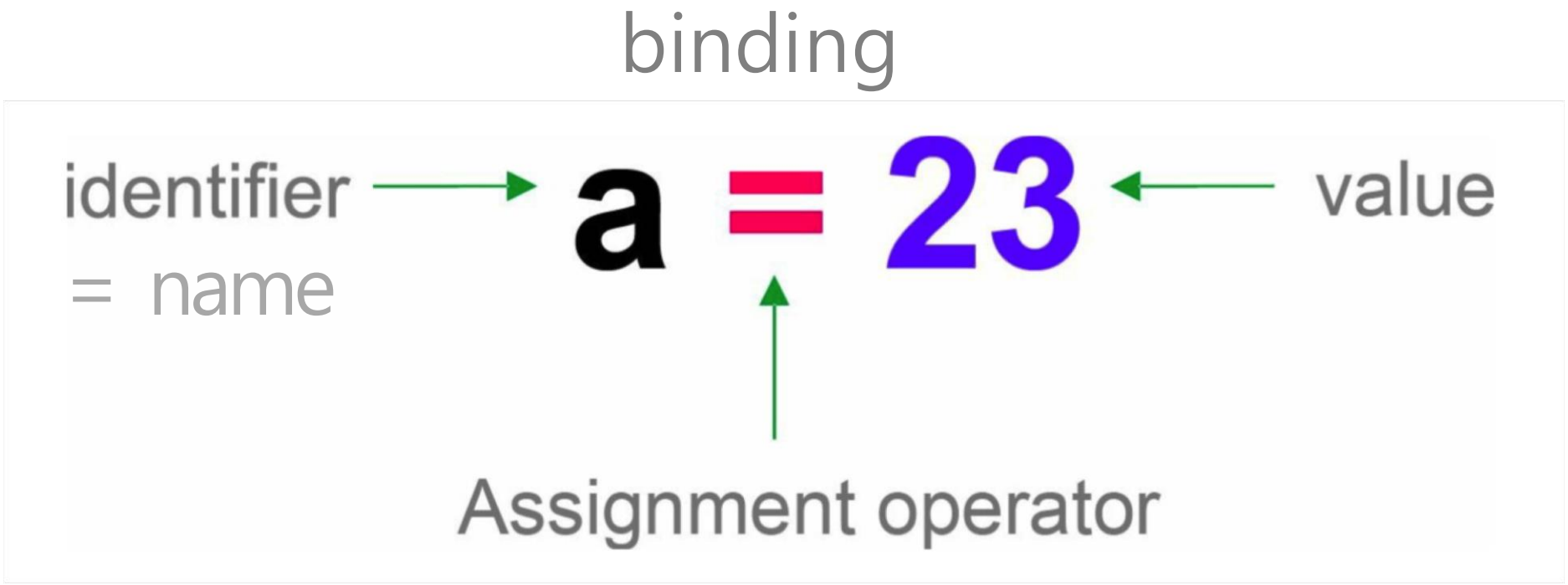
- 문자열 :텍스트의 묶음
- 문자열은 홑 따옴표 또는 쌍 따옴표로 묶어준다.
  - 예: `'Good', "good"`
- 문자열 + 문자열
  - 문자열도 숫자처럼 덧셈 연산으로 더할 수 있음
  - 여기서, 문자열의 덧셈은 문자열의 나열을 의미함

```
In [ ] : 'Hello' + 'World!'
Out[ ] :
```

```
In [ ] : 'Hello ' + 'World!'
Out[ ] :
```

Assignment

Assignment



■ 프로그래밍에서의 변수

○값을 기억해 두고 필요할 때 활용할 수 있음
- ➢중간 계산값을 저장하거나 누적 등

■ Python

○값(객체)을 저장하는 메모리 상의 공간을 가르키는(object reference) 이름

○Python은 모든 것이 객체이므로 변수보다는 식별자로 언급. 변수로 통용해서 사용하기도 함

○변수의 경우, 선언 및 할당이 동시에 이루어져야 함

○ case-sensitive names
- ○E.g., functions, classes and variables

- 변수(Variables)

  - = (대입 연산자)를 이용하여 값, 또는 연산 결과를 저장한다.
  - 예를 들어, 변수 spam에 값 15를 저장하려면

```
In [ ] : spam = 15
In [ ] : spam
Out[ ] :
```

  - 변수에 값을 저장하면 메모리에 저장하게 되는데, 메모리는 고유의 주소값을 가짐
  - 주소값을 확인하려면 id()함수를 사용

```
In [ ] : id(spam)
Out[ ] :
```

- %whos : 현재 메모리에 올라간 변수들의 목록을 확인

```
In [ ] : a = 10
         b = 'hi'
         c = 3.14
In [ ] : %whos

Variable   Type     Data/Info
-------------------------------
a          int      10
b          str      hi
c          float    3.14
```

- 변수를 더 이상 사용하고 싶지 않다면 del

```
In [ ] : del a
In [ ] : a
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-3f786850e387> in <module>
----> 1 a

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

```
In [ ] : %whos
```

```
Variable   Type     Data/Info
-------------------------------
b          str      hi
c          float    3.14
```

- 변수 이름을 지을 때...
  - 의미 있는 이름을 사용(권장)
  - 소문자와 대문자는 서로 다르게 취급
  - 변수의 이름은 문자와 숫자, 밑줄(_)로 이루어짐
  - 변수의 이름 중간에 공백이 들어가면 안됨. 단어를 구분하려면 밑줄(_)을 사용함
  - 숫자로 시작할 수 없음
  - 키워드는 사용할 수 없음
  - 낙타체(권장)
    - 변수의 첫 글자는 소문자, 나머지 단어의 첫 글자는 대문자로 적는 방법
      - 예) myNewCar

- Quiz
  - 변수 이름으로 사용하기에 적절할까요?

```
sum
_count
number_of_pictures
King3
2nd_base
money#
while
```

- Quiz
  - 변수 이름으로 사용하기에 적절할까요?

sum	(O) 영문 알파벳 문자로 시작
_count	(O) 밑줄 문자로 시작할 수 있다.
number_of_pictures	(O) 중간에 밑줄 문자를 넣을 수 있다.
King3	(O) 맨 처음이 아니라면 숫자도 넣을 수 있다.
2nd_base	(X) 숫자로 시작할 수 없다.
money#	(X) #과 같은 기호는 사용할 수 없다.
while	(X) 예약어는 사용할 수 없다.

- Quiz

```
In [ ] : spam = 15
In [ ] : spam + 5
Out[ ] :
```

```
In [ ] : spam = 3
In [ ] : spam + 5
Out[ ] :
```

```
In [ ] : spam = 5 + 7
In [ ] : spam
Out[ ] :
```

```
In [ ] : spam = 15
In [ ] : spam + spam
Out[ ] :
```

```
In [ ] : spam - spam
Out[ ] :
```

- Quiz

```
In [ ] : spam = 15
In [ ] : spam = spam + 5
In [ ] : spam
Out[ ] :
```

```
In [ ] : spam = 15
In [ ] : spam = spam + 5
In [ ] : spam = spam + 5
In [ ] : spam = spam + 5
Out[ ] :
```

Object Reference

Object Reference

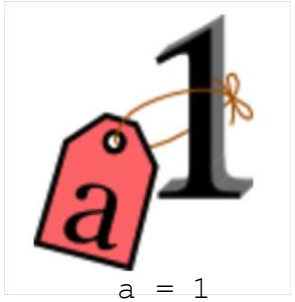
C


```
int a = 1;
```

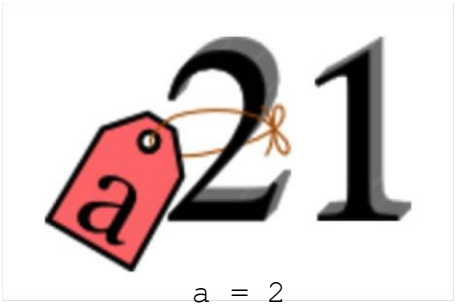

```
a = 2;
```


```
int b = a;
```

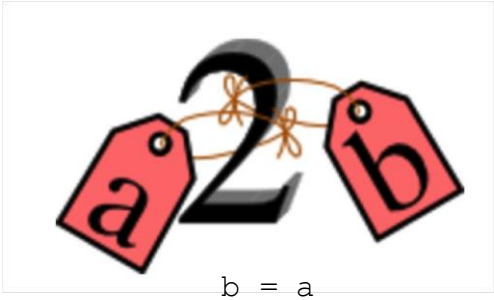
Python




```
a = 1
```

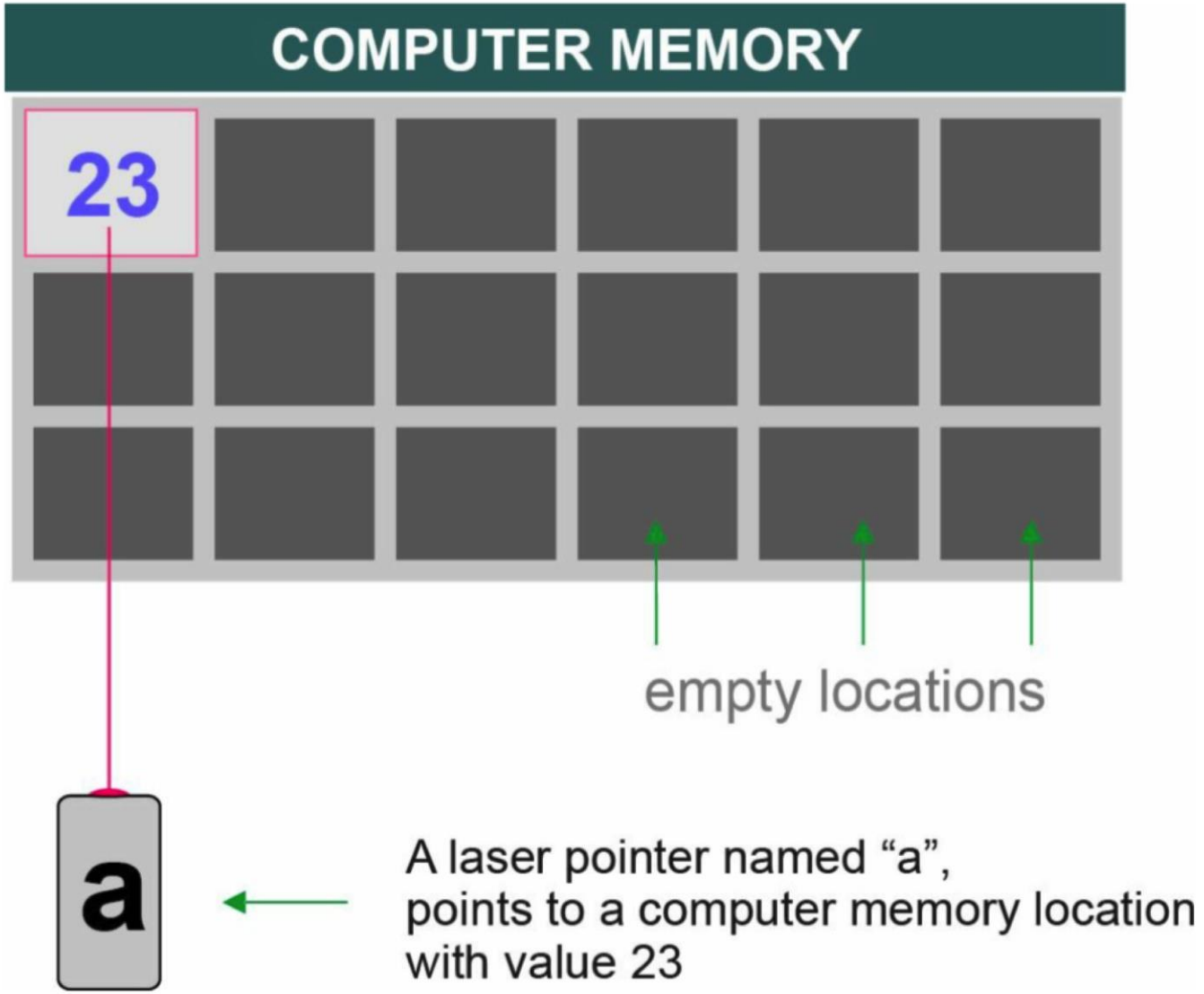



```
a = 2
```

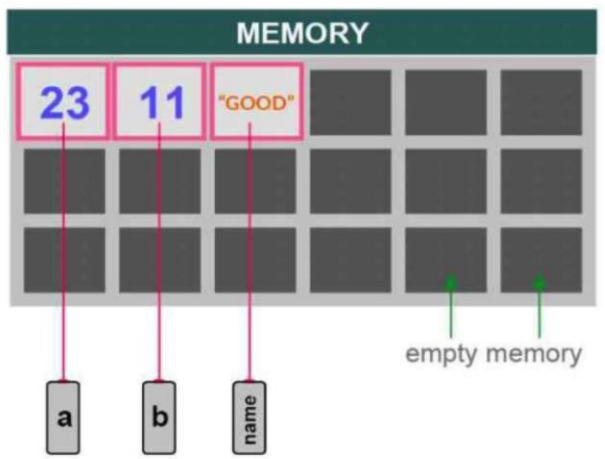



```
b = a
```

Object Reference

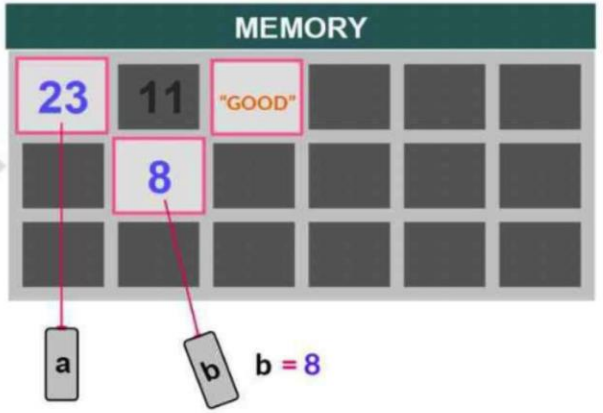


Object Reference

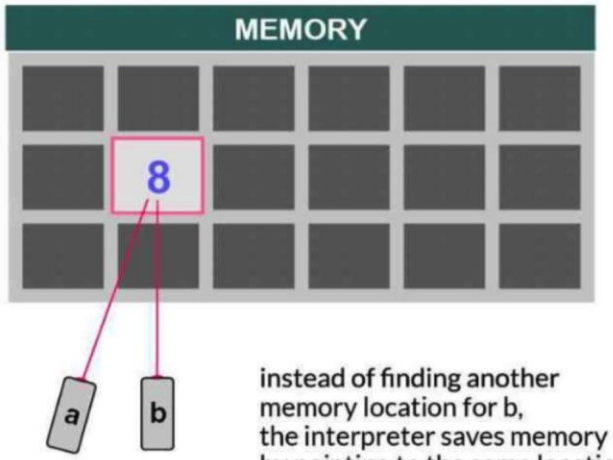


a = 23 b = 11 name ="GOOD"

lets change the value of b from 11 to 8

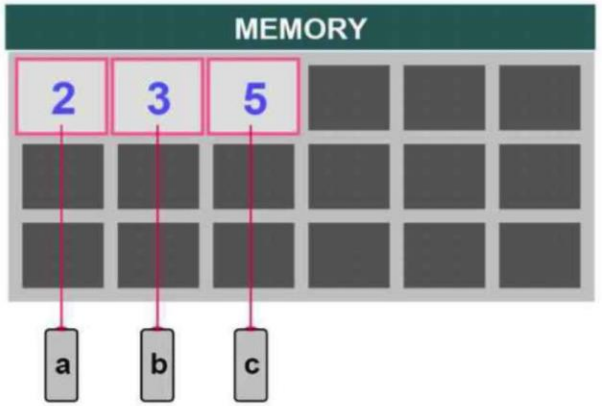


Because numbers are immutable, "b" changes location to the new value.
When there is no reference to a memory location the value fades away and the location is free to use again.
This process is known as garbage collection



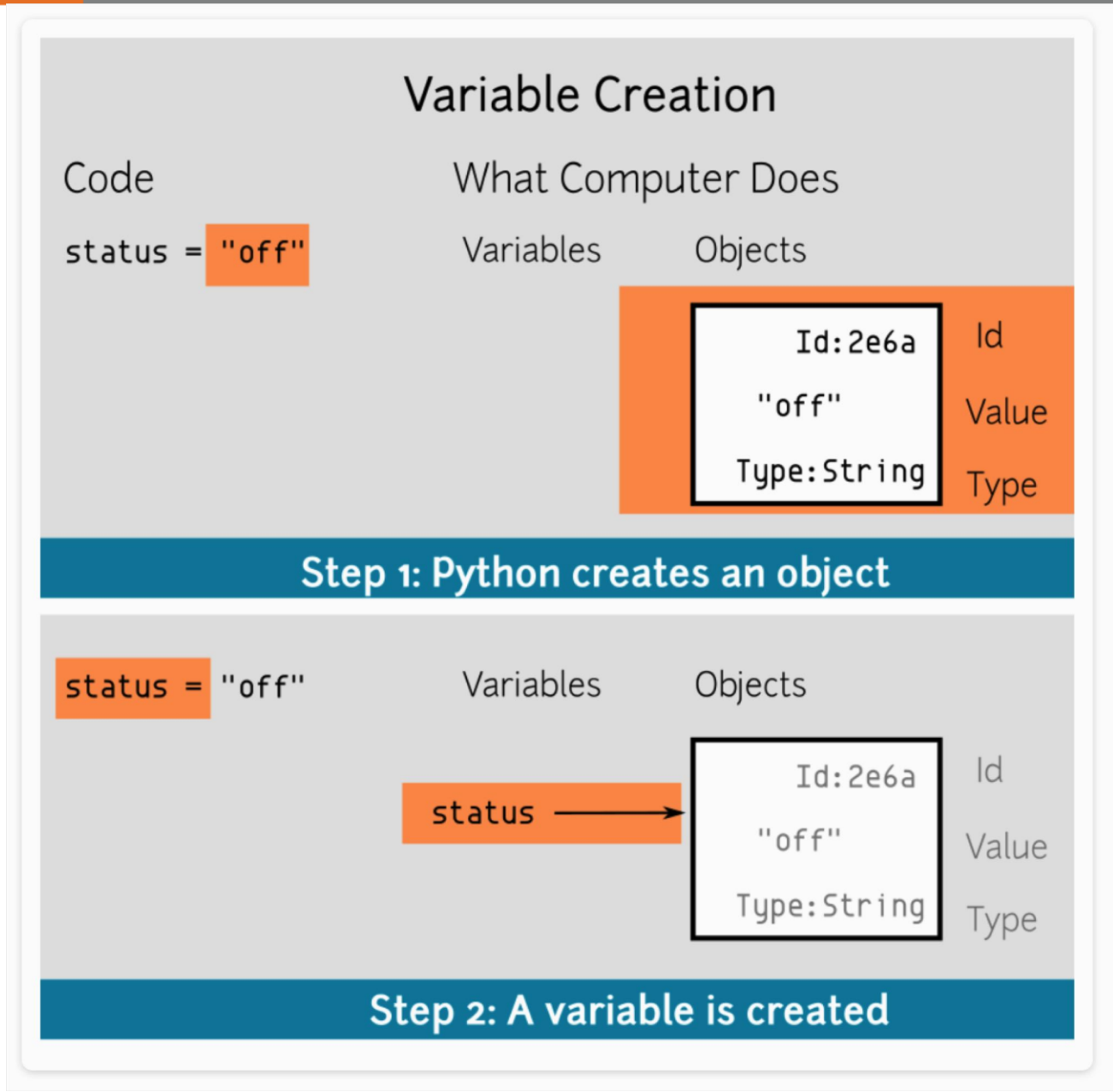
a = b

instead of finding another memory location for b, the interpreter saves memory by pointing to the same location

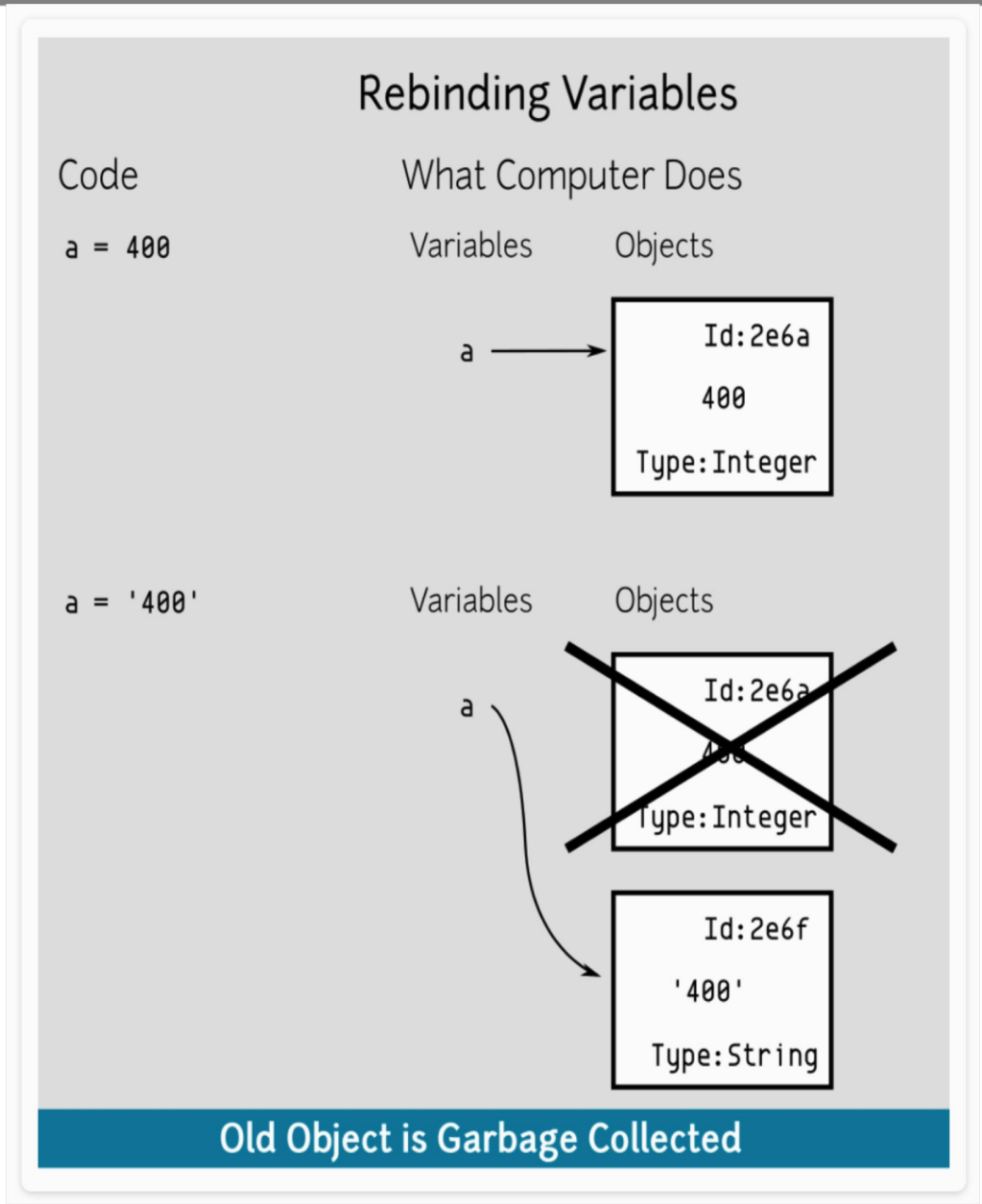


a = 2 b = 3 c = a + b

Object Reference



Object Reference



- 문자열
  - 문자열은 문자뿐만 아니라 공백, 숫자, 특수문자를 포함할 수 있다.

```
In [ ] : spam1 = 'hello'
In [ ] : spam2 = 'Hi there!'
In [ ] : spam3 = 'KITTENS'
In [ ] : spam4 = '7 apples, 14 oranges, 3 lemons'
In [ ] : spam5 = 'Anything not pertaining to elephants is irrelephant.'
In [ ] : spam6 = 'O*&#wY%*&OCfsdYO*& gfC%YO'
```

- Quiz

```
In [ ] : spam = 'Hello'
In [ ] : fizz = 'World!'
In [ ] : 'spam' + 'fizz'
Out[ ] :
```

```
In [ ] : spam = 'Hello'
In [ ] : fizz = 'World!'
In [ ] : spam + 'fizz'
Out[ ] :
```

```
In [ ] : spam = 'Hello'
In [ ] : fizz = 'World!'
In [ ] : 'hi' + spam + fizz
Out[ ] :
```

- Quiz

```
In [ ] : 'Hello' + '5'
Out[ ] :
In [ ] : '5' + '5'
Out[ ] :
```

```
In [ ] : spam = 5
In [ ] : 'Hello' + spam
Out[ ] :
```

```
In [ ] : spam = 'Hello'
In [ ] : spam = 'World!'
In [ ] : spam + spam
Out[ ] :
```

```
In [ ] : a=1
         a=b=c=a

In [ ] : lhs, rhs = lhs, rhs

In [ ] : counter = 0
         counter += 1

In [ ] : a, b = 1, 2

In [ ] : a

Out[ ] :

In [ ] : b

Out[ ] :

In [ ] : c

Out[ ] :
```

Comments

Comments (주석)

■ 주석

○해시 문자(#)로 시작하고 줄의 끝까지 이어집니다.

➢주석은 줄의 처음에서 시작할 수도 있고, 공백이나 코드 뒤에 나올 수도 있음

➢하지만 문자열 리터럴 안에는 들어갈 수 없습니다. 문자열 리터럴 안에 등장하는 해시 문자는 주석이 아니라 해시 문자일 뿐

➢주석은 코드의 의미를 정확히 전달하기 위한 것이고, 파이썬이 해석하지 않는 만큼, 예를 입력할 때는 생략해도 됨

■ #

○Line-based, i.e., but independent of indentation (but advisable to do so)

○There are no multi-line comments

➢''' '''

Comments (주석)


```
#This program says hello and asks for my name
print ('hello world!')
print('Input your name.')
myName = input()
print ('Welcome, '+ myName + '.')
```

```
hello world!
Input your name.
Haesun
Welcome, Haesun.
```

이렇게 줄 맨 앞에 #이 붙어 있으면
IDLE은 이 줄 전체를 그냥 무시

보통은, 이 코드를 볼 다른 친구,
또는 미래의 나를 위해
이 코드에 대한 설명을 달기 위해 사용

```
#This program says hello and asks for my name
print ('hello world!')
print('Input your name.')
```

```
hello world!
Input your name.
```

특정 코드를 잠시 실행시키고 싶지 않을 때도 사용

- python에서는 여러 줄 주석이 없음

print()

- **print( )**
  - print()는 모니터에 텍스트를 출력해주는 함수
  - 반드시 소문자만을 사용

- 소스코드

```
print ('Hello world!')
print ('Input your name.')
```

- 실행결과

```
Hello world!
Input your name.
>>>
```

- **print( )**
  - 줄 바꾸지 않기
    - print( ) 문자열 끝에 **end=' '**를 붙이면 **줄을 바꾸지 않음**
    - **end=' '**에서 **' '**안에 문자열을 넣어주면 해당 문자열이 줄의 끝에 삽입됨
      - 예) **' '**안에 공백을 넣으면 **줄을 바꾸지 않고** 공백만 추가, **'^^'** 쓰면 줄을 바꾸지않고 **^^**를 추가

• 소스코드

```
print ('Hello world! ', end=' ')
print ('Input your name.')
```

• 실행결과

```
Hello world! Input your
    name.
>>>
```

• 소스코드

```
print ('Hello world!', end='^^')
print ('Input your name.')
```

• 실행결과

```
Hello world!^^Input your name.
>>>
```

- **print( )**
  - 줄 바꾸기
    - **print( )**만 입력하면 **줄바꿈**

• 소스코드

```
print ('Hello world!')
print ( )
print ('Input your name.')
```

• 실행결과

```
Hello world!

Input your name.
>>>
```

- 문자열의 끝에 **₩n**을 넣으면 줄 바꿈(영문 글꼴의 경우 **\n**)

• 소스코드

```
print ('Hello world!₩nInput your name.')
```

• 실행결과

```
Hello world!
Input your name.
>>>
```

- print( )
  - 변수에 저장되어 있는 데이터 타입에 상관없이, 연산이 아닌 단순히 나열하여 결과를 출력하려면 ,(컴마)로 나열하는게 편함

• 소스코드

```
age = 25
print ('나이 ' + age)
name = "슈가"
print ('Nice to meet you, ' + name)
```

• 실행결과

```
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    print ('나이'+age)
TypeError: must be str, not int
>>>
```

• 소스코드

```
age = 25
print ('나이 ' , age)
name = "슈가"
print ('Nice to meet you, ' + name)
```

• 실행결과

```
나이 25
Nice to meet you, 슈가
>>>
```

- print( )
  - 변수에 저장되어 있는 데이터 타입에 상관없이, 연산이 아닌 단순히 나열하여 결과를 출력하려면 ,(컴마)로 나열하는게 편함

• 소스코드

```
age = 25
print ('나이 ' + age)
name = "슈가"
print ('Nice to meet you, ' + name)
```

더하기 연산하려면 문자인지 숫자인지 생각해야 해요.

• 실행결과

```
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    print ('나이'+age)
TypeError: must be str, not int
>>>
```

• 소스코드

```
age = 25
print ('나이 ', age)
name = "슈가"
print ('Nice to meet you, ' + name)
```

문자이든 숫자이든 상관없어요.

• 실행결과

```
나이 25
Nice to meet you, 슈가
>>>
```

- **print( )**
  - 연산과 나열의 예

- 소스코드

```
age = 25
print ('나이 ' , age)
name = "슈가"
print (' Nice to meet you, ' , name)
print (' 10년 후 나이는 ' , age+10 , '이군요. ')
```

연산이 필요한 경우에는 반드시 연산자를 써주어야 해요

- 실행결과

```
나이 25
Nice to meet you, 슈가
10년 후 나이는 35 이군요.
>>>
```

print()

- print('문자열 ' * 변수) 함수를 이용하여 다음의 실행 결과가 나오도록 소스코드를 작성해보세요.

실행결과

```
콩 콩 콩 콩 콩 콩 콩 콩 콩 콩 나무를 10번 찍었습니다.
>>>
```

단계 : 문제 분해

1. 글자 '콩 '이 10번 나왔으니 변수 num에 **10을** 저장한다.
2. 글자 '콩 '을 10번 출력하기 위해서는 **print('문자열 ' * 변수)** 하면 된다.
3. 나머지 글자들도 **print() 괄호 안에 포함시켜 나열**하여 출력한다.

print()

- 솔루션

```

```

- 오답은 아니지만 바람직하지 않는 예

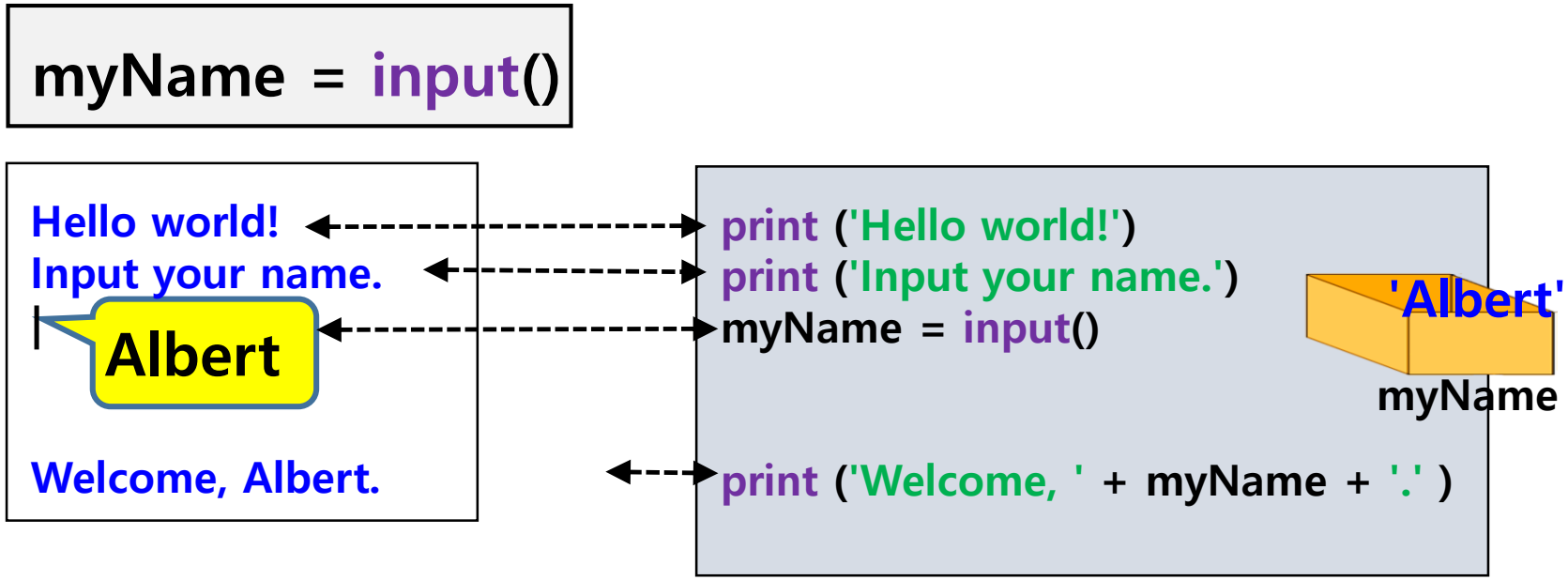
```
')
```

```

```

```

```

input()

- input()
  - **프로그램 실행 중에** 키보드로부터 데이터를 입력 받고자 할 때 사용하는 함수
  - 프로그램에서 input()을 만나면 쉘 윈도우에서 커서가 깜박거리면서 입력을 기다림
  - 입력 받은 데이터는 일반적으로 변수에 저장함
    - 변수 없이 input()만 선언하면 데이터는 입력 받지만 저장하지는 않음



- input()
  - ❖ 입력 받은 값은 **문자열(string)타입**!!!

• 소스코드

```
print ('How old are you?')
age = input()
print ('You are '  + age + ' years
old.')
```

• 실행결과

```
How old are you?
15
You are 15 years old.
>>>
```

- input()
  - input() 괄호 안에 문자열, 또는 변수를 넣어주면, 괄호 안에 데이터를 먼저 출력하고 입력을 받음
  - 두 줄의 코드를 한 줄로 줄여서 작성하는 것과 같은 의미

```
print ('How old are you?')
age = input()
```

실행순서

```
age = input('How old are you?')
```
③ ② ①

한가지 차이점이 있어요.

- 실행결과

```
How old are you?
      15
>>>
```

input()

```
name= 'Alice'
age = input(name+':')
```

괄호 안에 변수를 쓸 수도 있어요~

- 실행결과

```
Alice
>>>
```

```
name= 'Alice'
age = input(name, ':')
```

+가 아니라 ,를 쓰는 경우는 어떨까요?

- 실행결과

```
>>>
```

- Quiz

```
>>> a = input() + '5'
100
>>> a

>>> a = input() + 'Hello'
슈가
>>> a

>>> a = input() + 100
100
```

```
>>> number = input('input number: ')
input number: 100
>>> number = number + 100

>>> a = 5
>>> input(a)
```

input()

```
print ('Tell me your favorite stuff.')
fruit = input('Your favorite fruit is ')
flower = input('Your favorite flower is ')
drink = input('Your favorite drink is ')
print ('You entered: '  + fruit + ' ' + flower + '  ' + drink )
```

- 말 따라하는 앵무새 프로그램을 작성해보세요.

  - input()으로 입력한 데이터를 변수에 저장했다가 print()로 출력하도록 한다.

- **실행결과**

```
>>>
대화를 시작합니다.

너 이름이 뭐야?
너 이름이 뭐야?
>>>
```

검정색 글은 **사용자**가 입력하는 글

파랑색 글은 **프로그램**에 의해서 출력되는 글

- 앵무새 프로그램을 수정해보세요.
  - **'조련사:', '앵무새:'를 각각 변수에 저장해 두고 활용한다.**

- 실행결과

```
>>>
대화를 시작합니다.

조련사: 너 이름이 뭐야?
앵무새: 너 이름이 뭐야?
>>>
```

앵무새 프로그램3

- 앵무새 프로그램을 수정해보세요.
  - **3번의 문장을 따라하도록 한다. 동일한 소스코드를 3번 copy & paste**

- 실행결과

```
>>>
대화를 시작합니다.

조련사: 너 이름이 뭐야?
앵무새: 너 이름이 뭐야?

조련사: 난 블랙핑크 로제야.
앵무새: 난 블랙핑크 로제야.

조련사: 난 뉴욕을 매료시킨 완벽한 인형미모야.
앵무새: 난 뉴욕을 매료시킨 완벽한 인형미모야.

>>>
```

Value

■ 값에 대한 type이 중요함

■ 값에 따라 서로 다른 기술적인 체계가 필요

○ 지원하는 연산 및 기능이 다르기 때문

■ 컴퓨터에서는 이진수를 사용해서 값을 표현하고 관리

○ 숫자형 = numeric

- 산술 연산을 적용할수 있는 값
- 정수 : 0, 1, -1 과 같이 소수점 이하 자리가 없는 수. 수학에서의 정수 개념과 동일. (int)
- 부동소수 : 0.1, 0.5 와 같이 소수점 아래로 숫자가 있는 수. (float)
- 복소수 : Python에서 기본적으로 지원

○ 문자, 문자열

- 숫자 "1", "a", "A" 와 같이 하나의 낱자를 문자라 하며, 이러한 문자들이 1개 이상있는 단어/문장와 같은 텍스트
- Python에서는 낱자와 문자열 사이에 구분이 없이 기본적으로 str 타입을 적용
  - byte, bytearry

○ 불리언 = boolean

- 참/거짓을 뜻하는 대수값. 보통 컴퓨터는 0을 거짓, 0이 아닌 것을 참으로 구분
- True 와 False 의 두 멤버만 존재 (bool)
- Python에서는 숫자형의 일부

Type

○ Compound = Container = Collection

- 기본적인 데이터 타입을 조합하여, 여러 개의 값을 하나의 단위로 묶어서 다루는 데이터 타입
- 논리적으로 이들은 데이터 타입인 동시에 데이터의 구조(흔히 말하는 자료 구조)의 한 종류. 보통 다른 데이터들을 원소로 하는 집합처럼 생각되는 타입들
- Sequence
  - list : 순서가 있는 원소들의 묶음
  - tuple : 순서가 있는 원소들의 묶음. 리스트와 혼동하기 쉬운데 단순히 하나 이상의 값을 묶어서 하나로 취급하는 용도로 사용
  - range
- Lookup
  - mapping
    - » dict : 그룹내의 고유한 이름인 키와 그 키에 대응하는 값으로 이루어지는 키값 쌍(key-value pair)들의 집합.
  - set : 순서가 없는 고유한 원소들의 집합.

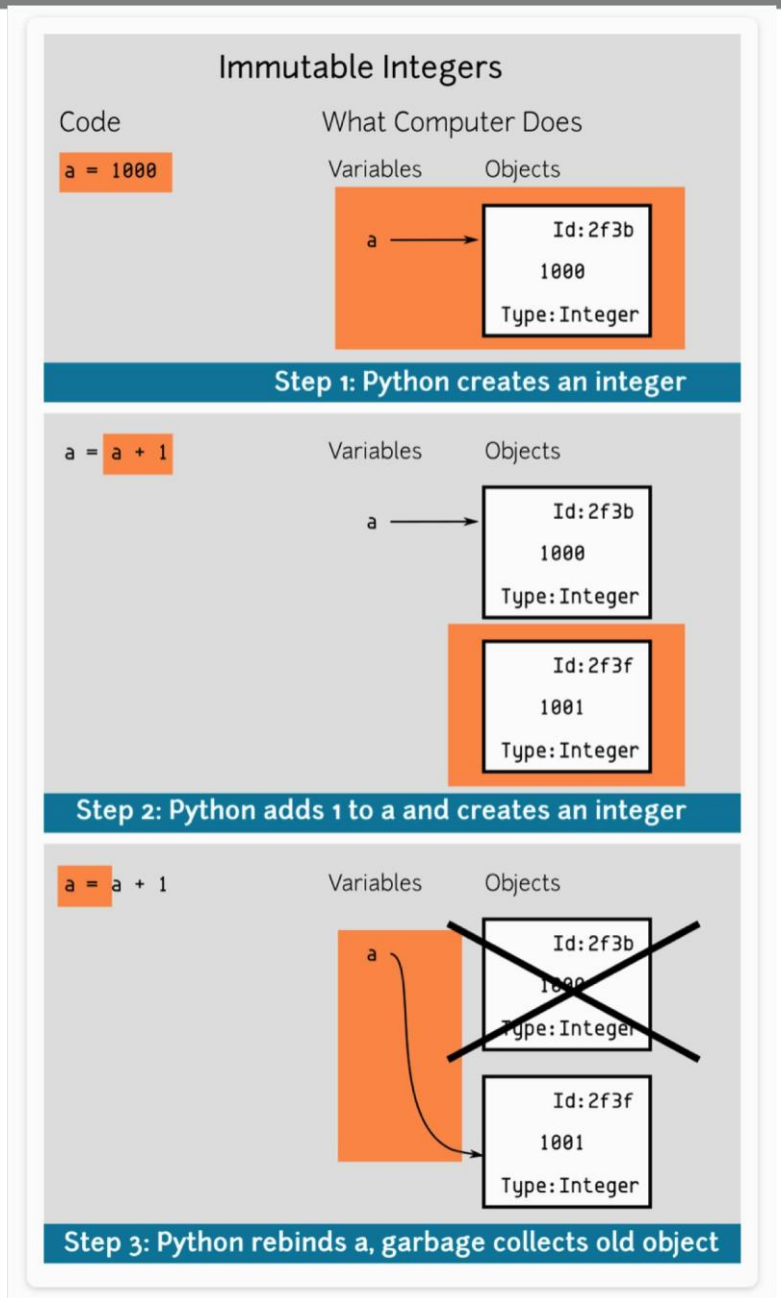
○ None

- 존재하지 않음을 표현하기 위해서 "아무것도 아닌 것"을 나타내는 값
- 어떤 값이 없는 상태를 가리킬만한 표현이 마땅히 없기 때문에 "아무것도 없다"는 것으로 약속해놓은 어떤 값을 하나 만들어 놓은 것
- None 이라고 대문자로 시작하도록 쓰며, 실제 출력해보아도 아무것도 출력되지 않음.
- 값이 없지만 False 나 0 과는 다르기 때문에 어떤 값으로 초기화하기 어려운 경우에 쓰기도 함

immutable

1. Numeric

1. int
2. float
3. bool


Immutable Integers
Code
What Computer Does
a = 1000
Variables
Objects
a
Id:2f3b
1000
Type:Integer
Step 1: Python creates an integer
a = a + 1
Variables
Objects
a
Id:2f3b
1000
Type:Integer
Id:2f3f
1001
Type:Integer
Step 2: Python adds 1 to a and creates an integer
a = a + 1
Variables
Objects
a
Id:2f3b
Type:Integer
Id:2f3f
1001
Type:Integer
Step 3: Python rebinds a, garbage collects old object

Numeric

Type		예시
int	int	14, -14
	int (Hex)	0xe
	int (Octal)	0o16
	int (Binary)	0b1110
float	float	14.0, 0.5, .3, 1.
	float	1.4e1, 1.79e+308 1.8e-308
	infinity	case insensible float("inf"), "Inf", "INFINITY" 및 "iNfINity"는 모두 가능
	Not a Number	case insensible float('nan')
complex	complex	14+0j
bool	bool	True, False
Underscore (readability)		1_000

■ 수를 표현하는 기본 리터럴

○ 정수

- 0, 100, 123 과 같은 표현

○ 실수

- 중간에 소수점 0.1, 4.2, 3.13123 와 같은 식
- 0. 으로 시작하는 실수값에서는 흔히 앞에 시작하는 0 제외 가능. .5 는 0.5를 줄여쓴 표현

○ 부호를 나타내는 –, + 를 앞에 붙일 수 있다. (-1, +2.3 등)

■ 기본적으로 그냥 숫자만 사용하는 경우, 이는 10진법 값으로 해석.

○ 10진법외에도 2진법, 8진법, 16진법이 존재.

- 2진법 숫자는 0b로 시작(대소문자를 구분하지 않음)
- 8진법 숫자는 0o로 시작(대소문자를 구분하지 않음)
- 16진법숫자는 0x로 시작(대소문자를 구분하지 않음)

■ 참/거짓을 의미하는 부울대수값.

○ 자체가 키워드로 True/False를 사용하여 표현

```
In [ ] : a = 10
In [ ] : a
Out[ ] :
```

```
In [ ] : type(a)
Out[ ] :
```

```
In [ ] : b = -2
In [ ] : b
Out[ ] :
```

```
In [ ] : type(b)
Out[ ] :
```

```
In [ ] : c = int(10)
In [ ] : c
Out[ ] :
```

```
In [ ] : type(c)
Out[ ] :
```

```
In [ ] : d = int(-32)
In [ ] : d
Out[ ] :
```

```
In [ ] : type(d)
Out[ ] :
```

- Quiz

```
In [ ] : c = int('10')
In [ ] : c
Out[ ] :
```

```
In [ ] : type(c)
Out[ ] :
```

```
In [ ] : d = int('ten')
In [ ] : d
Out[ ] :
```

```
In [ ] : type(d)
Out[ ] :
```

- 정수형의 최대 범위

```
In [ ] : import sys
         sys.maxsize

Out[ ] :
```

  - python에서는 최대값을 넘어가면 메모리를 동적으로 늘려줌
  - 즉, python에서의 정수형은 오버플로우가 없음

```
In [ ] : 9223372036854775809

Out[ ] :
```

실수형(float)


```
In [ ] : a=1.2
In [ ] : a
Out[ ] :
```

```
In [ ] : type(a)
Out[ ] :
```

```
In [ ] : b=-5.
In [ ] : b
Out[ ] :
```

```
In [ ] : type(b)
Out[ ] :
```

```
In [ ] : c = float(3.14)
In [ ] : c
Out[ ] :
```

```
In [ ] : type(c)
Out[ ] :
```

```
In [ ] : d = 3.14-1
In [ ] : d
Out[ ] :
```

```
In [ ] : type(d)
Out[ ] :
```

float

- 실수형의 최대 범위

```
In [ ] : sys.float_info
Out[ ] :
```

  - python에서는 실수형의 최대 범위를 넘어가게 되면 inf(무한대)로 변함

```
In [ ] : 1.7976931348623157e+310
Out[ ] : inf

In [ ] : e=float('inf')
In [ ] : e
Out[ ] :
```

float

- 실수형은 계산방식이 다르기 때문에 유의해야함

```
In [ ] : 0.1+0.1+0.1 == 0.3
Out[ ] : False
```

```
In [ ] : 0.1+0.1+0.1
Out[ ] : 0.30000000000000004
```

```
In [ ] : 1.7976931348623157e+308 == 1.7976931348623157e+308+1
Out[ ] : True
```

  - 실수값은 어림값으로 계산하기 때문

float

- Quiz

```
In [ ] : c = float('10')
In [ ] : c
Out[ ] :
```

```
In [ ] : type(c)
Out[ ] :
```

```
In [ ] : d = float('10.0')
In [ ] : d
Out[ ] :
```

```
In [ ] : type(d)
Out[ ] :
```

온도변환 프로그램

- 아래와 같은 온도 변환 프로그램을 작성해보세요.

```
온도 변환 프로그램
화씨 온도를 입력하세요:50
섭씨 온도는 10.0 입니다.

섭씨 온도를 입력하세요:50
화씨 온도는 122.0 입니다.

>>>
```

공식 :

화씨(f) -> 섭씨(c) c= 5.0/9.0*(f-32.0)

섭씨(c) -> 화씨(f) f=9.0/5.0*c+32.0

- 소스코드

BMI변환 프로그램

- 체중과 신장을 입력 받아 BMI(신체질량지수)를 계산하여 출력하는 프로그램을 작성하시오.
  - BMI 계산식 : 체중(kg) / (신장(cm)/100)^2

- 실행결과

```
체중을 입력하세요(kg): 48.4
신장을 입력하세요(cm): 178.7

BMI는 15.16 입니다.
```

- 소스코드

- Tip : 소수점 두자리까지만 출력하는 방법

```
print ('BMI는 %.2f'%BMI, '입니다')
```

변수 BMI의 값에서 소수점 두 자리 까지만 출력하라는 뜻

- 출력결과는 항상 int형

```
In [ ] : a = 0b010011
In [ ] : a
Out[ ] :  19
```

```
In [ ] : type(a)
Out[ ] :  int
```

```
In [ ] : b = bin(19)
In [ ] : b
Out[ ] : '0b10011'
```

```
In [ ] : type(b)
Out[ ] :  str
```

- 출력결과는 항상 int형

```
In [ ] : a = 0o724
In [ ] : a
Out[ ] :  468
```

```
In [ ] : type(a)
Out[ ] :  int
```

```
In [ ] : b = oct(468)
In [ ] : b
Out[ ] : '0o724'
```

```
In [ ] : type(b)
Out[ ] :  str
```

```
In [ ] : a = 0x19AF
In [ ] : a
Out[ ] :  6575
```

```
In [ ] : type(a)
Out[ ] :  int
```

```
In [ ] : b = hex(6575)
In [ ] : b
Out[ ] : '0o19af'
```

```
In [ ] : type(b)
Out[ ] :  str
```

논리(Boolean)형

```
In [ ] : a = True
In [ ] : a
Out[ ] :
```

```
In [ ] : type(a)
Out[ ] :  bool
```

```
In [ ] : a = bool(True)
In [ ] : a
Out[ ] :  True
```

```
In [ ] : type(a)
Out[ ] :  bool
```

```
In [ ] : b = False
In [ ] : b
Out[ ] : False
```

```
In [ ] : type(b)
Out[ ] :  bool
```

```
In [ ] : b = bool(False)
In [ ] : b
Out[ ] :  False
```

```
In [ ] : type(b)
Out[ ] :  bool
```

논리(Boolean)형

```
In [ ] : a+1
Out[ ] : 2
```

```
In [ ] : b+1
Out[ ] : 1
```

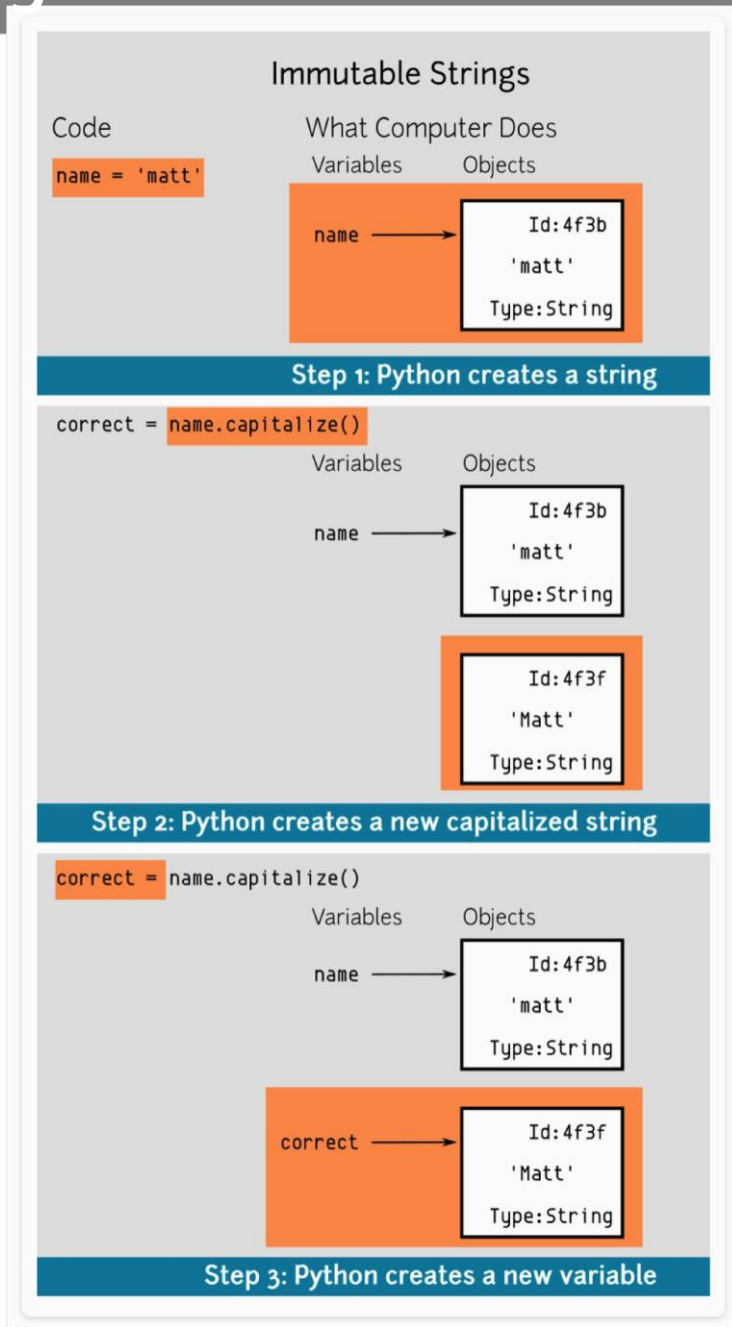
```
In [ ] : a==1
Out[ ] : True
```

```
In [ ] : b==0
Out[ ] : True
```

immutable, flat sequence

2. String

string


Immutable Strings
Code
What Computer Does
name = 'matt'
Variables
Objects
name
Id:4f3b
'matt'
Type:String
Step 1: Python creates a string
correct = name.capitalize()
Variables
Objects
name
Id:4f3b
'matt'
Type:String
Id:4f3f
'Matt'
Type:String
Step 2: Python creates a new capitalized string
correct = name.capitalize()
Variables
Objects
name
Id:4f3b
'matt'
Type:String
correct
Id:4f3f
'Matt'
Type:String
Step 3: Python creates a new variable

■ str

○글자, 글자가 모여서 만드는 단어, 문장, 여러 줄의 단락이나 글 전체

■ literal

○문자열은 홑따옴표, 쌍따옴표, doc스트링(''' ''')으로 표현

○큰 따옴표와 작은 따옴표를 구분하지 않지만 양쪽 따옴표가 맞아야 함. (문자열을 둘러싸는 따옴표와 다른 따옴표는 문자열 내의 일반 글자로 해석)

➢ "apple" , 'apple' 은 모두 문자열 리터럴로 apple이라는 단어를 표현

○두 개의 문자열 리터럴이 공백이나 줄바꿈으로 분리되어 있는 경우에 이것은 하나의 문자열 리터럴로 해석

➢ "apple," "banana"는 "apple,banana"라고 쓴 표현과 동일

- 따옴표 세 개를 연이어서 쓰는 경우에는 문자열 내에서 줄바꿈이 허용.
  - 흔히 함수나 모듈의 간단한 문서화 텍스트를 표현할 때 쓰임.

  - """He said "I didn't go to 'SCHOOL' yesterday"."""
    - He said "I didn't go to 'SCHOOL' yesterday".

  - 여러 줄에 대한 내용을 쓸 때.
  - '''HOMEWORK:

1. print "hello, world"
2. print even number between 2 and 12
3. calculate sum of prime numbers up to 100,000 ''

```
print ('안녕하세요')
print ("안녕하세요")
print ('''안녕하세요''')
```

```
안녕하세요
안녕하세요
안녕하세요
```

1

```
In [ ] : a="hello"
In [ ] : a
Out[ ] :
```

```
In [ ] : type(a)
Out[ ] :
```

2

```
In [ ] : b="안녕하세요"
In [ ] : b
Out[ ] :
```

```
In [ ] : type(b)
```

3

```
In [ ] : d="hello₩nworld!"
In [ ] : d
Out[ ] :
```

```
In [ ] : print(d)
hello
world!
```

4

```
In [ ] : e = r"hello₩nworld!"
In [ ] : e
Out[ ] : 'hello\\nworld!'
In [ ] : print(e)
hello₩nworld!
```

- 앞에 f를 붙여 사용하면 파이썬 코드의 실행결과를 문자열로 대입할 수 있음

```
In [1]: b={1==2}

In [2]: b
Out[2]: {False}

In [3]: type(b)
Out[3]: set

In [4]: b=1==2

In [5]: b
Out[5]: False

In [6]: type(b)
Out[6]: bool

In [7]: b=f'{1==2}'

In [8]: b
Out[8]: 'False'

In [9]: type(b)
Out[9]: str
```

- 변수로 한글사용 가능

```
In [13]: 일= 1

In [14]: 일
Out[14]: 1

In [15]: 이름='변해선'

In [16]: 이름
Out[16]: '변해선'

In [17]: type(이름)
Out[17]: str
```

Escape Sequence	Output
\newline	Ignore trailing newline in triple quoted string
\\	Backslash
\'	Single quote
\"	Double quote
\b	ASCII Backspace
\n	Newline
\r	ASCII carriage return
\t	Tab
\u12af	Unicode 16 bit
\U12af89bc	Unicode 32 bit
\N{BLACK STAR}	Unicode name
\o84	Octal character
\xFF	Hex character

print(...)

```
In [ ] : print('Single-quoted string' )
Out[ ] :
```

```
In [ ] : print("Double-quoted string" )
Out[ ] :
```

```
In [ ] : print('String with₩nnewline')
Out[ ] :
```

```
In [ ] : print('Unbroken₩
string')
Out[ ] :
```

```
In [ ] : print(r'₩n is an escape code')
Out[ ] :
```

```
In [ ] : print ("""String with
newline""")
Out[ ] :
```

- ASCII 코드값을 출력하려면,
  - ord(character)

```
In [34]: ord('a')
Out[34]: 97
```

- 문자열의 길이를 출력하려면,
  - len(string)

```
In [35]: len("hello")
Out[35]: 5
```

- 코드값을 가지고 해당 기호를 알아내려면
  - chr(codepoint)

```
In [37]: chr(10)
Out[37]: '
'
```

- 문자타입으로 변환하려면,
  - str(100)

```
In [38]: str(100)
Out[38]: '100'
```

Operation

Operation(연산)

■ 산술연산

○ 계산기

■ 비교연산

○ 동등 및 대소를 비교. 참고로 '대소'비교는 '전후'비교가 사실은 정확한 표현

○ 비교 연산은 숫자값 뿐만 아니라 문자열 등에 대해서도 적용할 수 있음.

■ 비트연산

■ 멤버십 연산

○ 특정한 집합에 어떤 멤버가 속해있는지를 판단하는 것으로 비교연산에 기반을 둠

○ is, is not : 값의 크기가 아닌 값 자체의 정체성(identity)이 완전히 동일한지를 검사

○ in, not in : 멤버십 연산. 어떠한 집합 내에 원소가 포함되는지를 검사 ('a' in 'apple')

■ 논리연산

○ 비교 연산의 결과는 보통 참/거짓. 이러한 불리언값은 다음의 연산을 적용. 참고로 불리언외의 타입의 값도 논리연산을 적용

연산자 우선 순위

	Operator	Description
lowest precedence	or	Boolean OR
	and	Boolean AND
	not	Boolean NOT
	in, not in	membership
	==, !=, <, <=, >, >=, is, is not	comparisons, identity
	|	bitwise OR
	^	bitwise XOR
	&	bitwise AND
	<<, >>	bit shifts
	+, -	addition, subtraction
	*, /, //, %	multiplication, division, floor division, modulo
highest precedence	+x, -x, ~x	unary positive, unary negation, bitwise negation
	**	exponentiation

PEMDAS :

Parentheses - Exponentiation - Multiplication - Division - Addition - Subtraction

Operation(연산)

■ ++ or -- 연산자는 없음

Arithmetic	Bitwise
+=	&=
-=	|=
*=	^=
/=	>>=
%=	<<=
//=	
**=	

조건문

■ 구문(statement) = 문

○ 예약어(reserved word, keyword)와 표현식을 결합한 패턴

○ 컴퓨터가 수행해야 하는 하나의 단일 작업(instruction)을 명시.

- ➢ 할당(대입, assigning statement)
  - – python에서는 보통 '바인딩(binding)'이라는 표현을 씀, 어떤 값에 이름을 붙이는 작업.
- ➢ 선언(정의, declaration)
  - – 재사용이 가능한 독립적인 단위를 정의. 별도의 선언 문법과 그 내용을 기술하는 블럭 혹은 블럭들로 구성.
    - » Ex) python에서는 함수나 클래스를 정의
  - – 블럭
    - » 여러 구문이 순서대로 나열된 덩어리
    - » 블럭은 여러 줄의 구문으로 구성되며, 블럭 내에서 구문은 위에서 아래로 쓰여진 순서대로 실행.
    - » 블럭은 분기문에서 조건에 따라 수행되어야 할 작업이나, 반복문에서 반복적으로 수행해야 하는 일련의 작업을 나타낼 때 사용하며, 클래스나 함수를 정의할 때에도 쓰임.
- ➢ 조건(분기) : 조건에 따라 수행할 작업을 나눌 때 사용.
  - – Ex) if 문
- ➢ 반복문 : 특정한 작업을 반복수행할때 사용.
  - – Ex) for 문 및 while 문
- ➢ 예외처리

- 조건문 형식 1

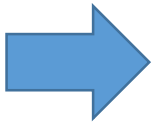
```
if 비교식:
    실행문장
```

조건문에는 세가지 형식이 있어요.

- 조건문 형식 1의 일반 예

파이썬으로 어떻게 작성할까요?

score가 60점 이상이면 조건
→ 합격입니다. 실행문장



- 조건문 형식1 예

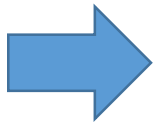
```
if score >= 60:
    print ( '합격입니다.' )
```

- 조건문 형식 2

```
if 비교식:
    실행문장
else :
    실행문장
```

- 조건문 형식 2의 일반 예

score가 60점 이상이면 조건
→ 합격입니다. 실행문장
그렇지 않으면 위의 조건이 아닐때
→ 불합격입니다. 실행문장

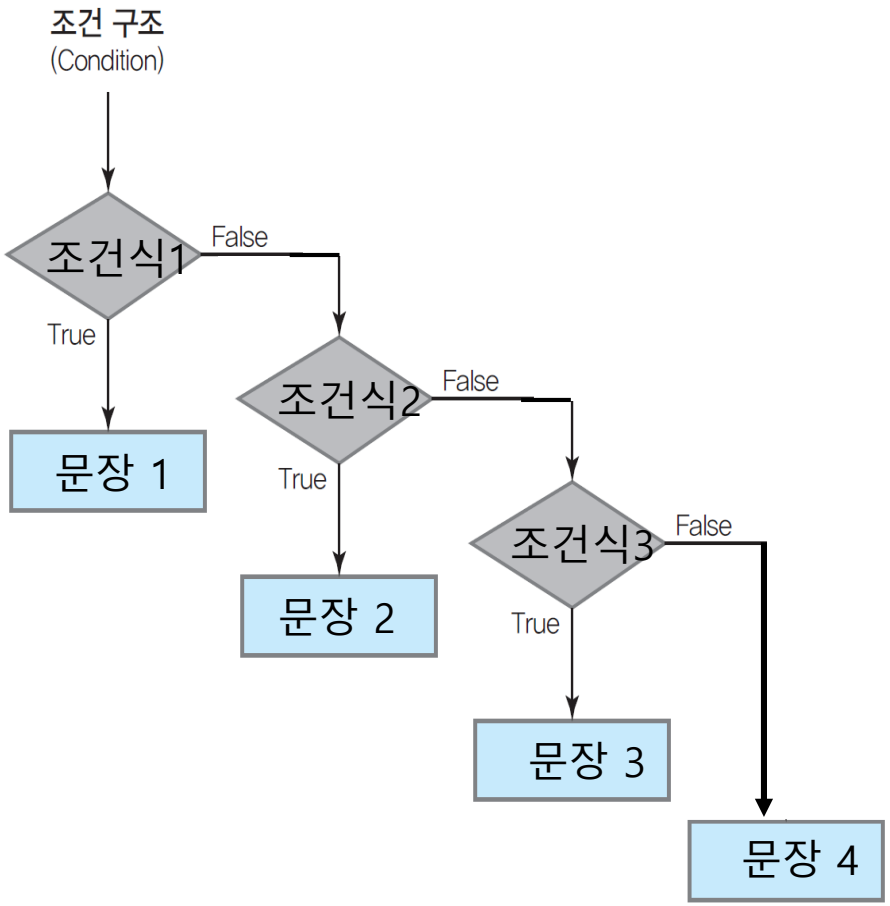


- 조건문 형식2 예

```
if score >= 60:
    print ('합격입니다. ' )
else :
    print ('불합격입니다.' )
```

- 조건문 형식 3

```
if 비교식1:
    실행문장1
elif 비교식2:
    실행문장2
elif 비교식3:
    실행문장3
else :
    실행문장4
```



- 조건문 형식3 예

```
if score >= 90:
    print ('A입니다. ' )
elif score >= 80 :
    print ('B입니다.' )
elif score >= 70 :
    print ('C입니다.' )
elif score >= 60 :
    print ('D입니다.' )
else:
    print ('F입니다.' )
```

- Quiz
  - 출력결과가 어떻게 다를까요?

```
score = 75
if score >= 90:
    print ('A입니다. ' )
elif score >= 80 :
    print ('B입니다.' )
elif score >= 70 :
    print ('C입니다.' )
elif score >= 60 :
    print ('D입니다.' )
else:
    print ('F입니다.' )
```

```
score = 75
if score >= 90:
    print ('A입니다. ' )
if score >= 80 :
    print ('B입니다.' )
if score >= 70 :
    print ('C입니다.' )
if score >= 60 :
    print ('D입니다.' )
else:
    print ('F입니다.' )
```

Indentation

- 들여쓰기의 중요성
  - 조건이 True일 때 실행해야 할 문장들을 블록으로 만들어 줘야 함
  - 블록 : 동일한 조건하에 실행되는 문장들을 묶어놓은 것
  - 동일 블록은 들여쓰기가 일치해야 함

```
if score > 90 :
    print ('합격입니다.')
    print ('A 등급입니다.')
```

블록: 동일한 조건하에 실행되어야 할 문장들을 묶은 것

Indentation

- 블록

```
if guess < number:
    print ('Your guess is too low.')

if guess > number:
    print ('Your guess is too high.')

if guess == number:
    print ('Good job, ' + myName + '! You guessed my number!')
    print ('I am happy.')
```

블록 1

블록 2

블록 3

Indentation

- 블록
  - 하나의 조건 아래 들여쓰기 길이가 다르면 다른 블록으로 간주하여 에러를 발생시킴

```
if score > 90 :
    print ('합격입니다.')
    print ('A 등급입니다.')
print ('본 과정을 수료하셨습니다.')
    print ('효도하시는군요!')
```

```
if area == '서울' :
    if gu == '성동구' :
        print (area, gu,  ' 명품교육의 도시')
    elif gu == '노원구' :
        print (area, gu, ' 힐링의 도시')
```

- 들여쓰기가 다른데 에러 날까요?

Indentation

- Quiz

- guess=10, number=12일때 예상되는 실행결과는?

```
if guess < number:
    print ('Your guess is too low.')

if guess > number:
    print ('Your guess is too high.')

if guess == number:
    print ('Good job, ' + myName + '! You guessed my number!‘)
print ('I am happy.')
```

문자열 비교


```
In [28]: 'ABC' > 'abc'
Out[28]: False

In [29]: 'z1Ab' > 'a3Bc'
Out[29]: True

In [30]: 'z'>'A'
Out[30]: True

In [31]: '123' > '96'
Out[31]: False

In [33]: '123' < '124'
Out[33]: True
```

비교연산자

- 비교연산자 종류

>	<	>=	<=	==	!=
크다	작다	크거나 같다	작거나 같다	같다	같지않다

- 비교의 결과는 Boolean type : True or False

```
>>> 0 < 6
True
>>> 0 > 6
False
>>> 10 < 10
False
>>> 10 == 10
True
```

비교연산자

- Quiz

```
>>> 11 >= 10
>>> 11 => 10

>>> 10 <= 11
>>> 10 =< 11

>>> 11 == 11
>>> 11 = 11
```

```
>>> 'Hello' == 'Hello'
>>> 'Hello' == 'HELLO'

>>> 5 == 5
>>> 5 == '5'

>>> 5 == int('5')
```

큰 수/작은 수/같은 수 출력하기

- 아래와 같이 두 수를 입력 받아 큰 수, 작은 수, 같은 수 인지를 출력하는 프로그램을 추가하세요.
  - if문과 비교연산자, int()를 사용하여 작성해보세요.

```
첫 번째 정수를 입력하시오: 100
두 번째 정수를 입력하시오: 200

큰수: 200
작은수: 100
>>>
첫 번째 정수를 입력하시오: 56
두 번째 정수를 입력하시오: 24

큰수: 56
작은수: 24
>>>
첫 번째 정수를 입력하시오: 45
두 번째 정수를 입력하시오: 45

두 수는 같습니다.
>>>
```

합격 여부 판단하는 프로그램

- 아래와 같이 성적을 입력 받아 90점 이상이면 합격, 90점 미만이면 불합격 여부를 판단하는 프로그램을 작성하시오.

```
점수를 입력하시오.
97
합격입니다.
>>>
점수를 입력하시오.
76
불합격입니다.
>>>
```

Ln: 3 Col: 4

짝수 홀수 검사하는 프로그램

- 아래와 같이 숫자를 입력 받아 짝수인지 홀수인지 검사하는 프로그램을 나머지 연산자(%)를 이용하여 작성하시오.

```
숫자를 입력하시오.
97
홀수입니다.
>>>
숫자를 입력하시오.
76
짝수입니다.
>>>
```

Ln: 3 Col: 4