IPA 주관 인공지능센터 기본(fundamental) 과정

- GitHub link: here
- E-Mail: windkyle7@gmail.com

Python Type and Identifier

파이썬에서 모든 자료형은 객체(object)이며 다른 프로그래밍 언어들과 달리 식별자 앞에 자료형을 명시해주지 않는다. 파이썬에서는 자료형을 크게 두 가지로 나눌 수 있다. 수를 표현하는 숫자형(Numeric)과 자료구조형 컨테이너(Container)가 있다. 그 중 변할 수 있는 가변(Mutable) 자료형, 변할 수 없는 불변(Immutable) 자료형으로 또다시 나눌 수 있다. 가변 자료형은 요소를 추가, 삭제, 수정이 가능하지만 불변 자료형은 한번 선언이 되면 추가, 삭제, 수정이 불가능하다. 또한 각 자료형마다 순서가 있는(Sequence) 자료형과 순서가 없는(Non-sequence) 자료형으로 더 나눌 수 있다.

식별자 (Identifier)

일반적으로 프로그래밍 언어에서 값을 저장하는 공간을 `변수(Value)`라 한다. 그러나 파이썬에서는 그 의미가 조금 다른 `식별자(Identifier)`라 부른다.

In [1]:
```
a = 10
```

위 코드에서 a를 식별자라고 한다. 이렇게 선언된 값들은 메모리에 올라가게 된다. 메모리에는 고유의 주소값을 가지고 있는데, 그 주소값을 확인하려면 다음과 같이 `id()` 함수를 사용하여 알 수 있다.

In [2]:
```
id(a)
```

Out[2]:
```
10910688
```

출력되는 값은 a에 대한 메모리 상의 주소값이며, 다른 컴퓨터에서 실행하거나 재할당하거나 다시 실행할 때마다 다를 수 있다.

현재 메모리에 올라간 식별자들의 목록을 확인해보고 싶다면 아래와 같이 jupyter notebook의 매직 키워드인 `%whos`를 사용해서 출력할 수 있다.

In [3]:
```
a = 10
b = 'hi'
c = 3.14
```

In [4]:
```
%whos
```

```
Variable          Type                          Data/Info
------------------------------------------------------------
NamespaceMagics   MetaHasTraits                 <class 'IPython.core.magi<...>mespace.NamespaceMagics'>
a                 int                           10
autopep8          module                        <module 'autopep8' from '<...>te-packages/autopep8.py'>
b                 str                           hi
c                 float                         3.14
get_ipython       function                      <function get_ipython at 0x7ffbd5be9d08>
getsizeof         builtin_function_or_method    <built-in function getsizeof>
json              module                        <module 'json' from '/usr<...>hon3.6/json/__init__.py'>
np                module                        <module 'numpy' from '/ho<...>kages/numpy/__init__.py'>
var_dic_list      function                      <function var_dic_list at 0x7ffb92b39a60>
yapf_reformat     function                      <function yapf_reformat at 0x7ffbcc199950>
```

파이썬에서는 각 식별자들은 현재 메모리 주소를 가르키고 있으므로 값을 저장하는 공간의 개념이 아닌 값 대신 부르는 이름이라는 개념에 더 가깝다.

식별자를 더 이상 사용하고 싶지 않다면 아래와 같이 `del` 이라는 키워드를 입력해준다.

In [5]:
```
del a
```

a를 셀에 입력하여 출력하려고 하면 더 이상 식별자로 정의되어 있지 않으므로 `NameError`가 발생한다.

In [6]:
```
a
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-3f786850e387> in <module>
----> 1 a

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

`%whos`로 다시 한번 확인해보면 a 식별자가 없어진 것을 확인할 수 있다.

In [7]:
```
%whos
```

```
Variable          Type                          Data/Info
------------------------------------------------------------
NamespaceMagics   MetaHasTraits                 <class 'IPython.core.magi<...>mespace.NamespaceMagics'>
autopep8          module                        <module 'autopep8' from '<...>te-packages/autopep8.py'>
b                 str                           hi
c                 float                         3.14
get_ipython       function                      <function get_ipython at 0x7ffbd5be9d08>
getsizeof         builtin_function_or_method    <built-in function getsizeof>
json              module                        <module 'json' from '/usr<...>hon3.6/json/__init__.py'>
np                module                        <module 'numpy' from '/ho<...>kages/numpy/__init__.py'>
```

```
var_dic_list      function                  <function var_dic_list at 0x7ffb92b39a60>
yapf_reformat     function                  <function yapf_reformat at 0x7ffbcc199950>
```

Type

- 값에 대한 type이 중요함
- 값에 따라 서로 다른 기술적인 체계가 필요
  - 지원하는 연산 및 기능이 다르기 때문
- 컴퓨터에서는 이진수를 사용해서 값을 표현하고 관리
  - 정확하게 표현하지 못할 수가 있음
- 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__
- 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 원 소로 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

숫자형 (Numeric)

정수형 (int)

```
In [8]: a = 10

In [9]: a
Out[9]: 10

In [10]: type(a)
Out[10]: int

In [11]: b = -2

In [12]: b
Out[12]: -2

In [13]: type(b)
Out[13]: int

In [14]: c = int(10)

In [15]: c
Out[15]: 10

In [16]: type(c)
Out[16]: int

In [17]: d = int(-32)

In [18]: d
Out[18]: -32

In [19]: type(d)
Out[19]: int
```

정수형의 최대 범위

In [20]:
```
import sys
sys.maxsize
```

Out[20]:
```
9223372036854775807
```

파이썬은 최대값을 넘어가면 메모리를 동적으로 늘려준다. 즉, 파이썬에서의 정수형은 오버플로우가 없다.

In [21]:
```
9223372036854775809
```

Out[21]:
```
9223372036854775809
```

실수형 (float)

In [22]:
```
a = 1.2
```

In [23]:
```
a
```

Out[23]:
```
1.2
```

In [24]:
```
type(a)
```

Out[24]:
```
float
```

In [25]:
```
b = -5.
```

In [26]:
```
b
```

Out[26]:
```
-5.0
```

In [27]:
```
type(b)
```

Out[27]:
```
float
```

In [28]:
```
c = float(3.14)
```

In [29]:
```
c
```

Out[29]:
```
3.14
```

In [30]:
```
type(c)
```

Out[30]:
```
float
```

In [31]:
```
d = 3.14 - 1
```

In [32]:
```
d
```

Out[32]:
```
2.14
```

In [33]:
```
type(d)
```

Out[33]:
```
float
```

실수형의 최대 범위

파이썬에서는 실수형의 최대 범위를 넘어가게 되면 `inf(무한대)` 로 변한다.

In [34]:
```
sys.float_info
```

Out[34]:
```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021, min_10_
exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

In [35]:
```
1.7976931348623157e+310
```

Out[35]:
```
inf
```

In [36]:
```
e = float('inf')
```

In [37]:
```
e
```

Out[37]:
```
inf
```

실수형은 계산 방식이 다르기 때문에 유의해야한다.

In [38]:
```
0.1 + 0.1 + 0.1 == 0.3
```

Out[38]:
```
False
```

In [39]:
```
0.1 + 0.1 + 0.1
```

Out[39]:
```
0.30000000000000004
```

In [40]:
```
1.7976931348623157e+308 == 1.7976931348623157e+308 + 1
```

Out[40]:
```
True
```

실수값은 계산 시 범위가 정해져 있으므로 정확히 알 수 없다. 따라서 어림값으로 계산하기 때문에 실수 계산 시 이 점을 유의해야 한다.

복소수 (complex)

In [41]:
```
a = 3 + 1j
```

In [42]:
```
a
```

Out[42]: (3+1j)

In [43]:
```
type(a)
```

Out[43]: complex

In [44]:
```
b = -1 - 4j
```

In [45]:
```
b
```

Out[45]: (-1-4j)

In [46]:
```
type(b)
```

Out[46]: complex

In [47]:
```
c = complex(1, 4j)
```

In [48]:
```
c
```

Out[48]: (-3+0j)

In [49]:
```
type(c)
```

Out[49]: complex

In [50]:
```
d = complex(-3 + 4j)
```

In [51]:
```
d
```

Out[51]: (-3+4j)

In [52]:
```
type(d)
```

Out[52]: complex

진법 표현

2진수

In [53]:
```
a = 0b010011
```

출력되는 결과는 항상 int형으로 나온다.

In [54]:
```
a
```

Out[54]: 19

In [55]:
```
type(a)
```

Out[55]: int

생성자 방식으로 10진수를 선언한 경우, 그에 상응하는 진법으로 바꾸어 문자열로 결과가 나타난다.

In [56]:
```
b = bin(19)
```

In [57]:
```
b
```

Out[57]: '0b10011'

In [58]:
```
type(b)
```

Out[58]: str

8진수

In [59]:
```
a = 0o724
```

In [60]:
```
a
```

Out[60]: 468

In [61]:
```
type(a)
```

Out[61]: int

In [62]:
```
b = oct(468)
```

In [63]:
```
b  # 결과는 문자열로 나온다.
```

Out[63]:
```
'0o724'
```

In [64]:
```
type(b)
```

Out[64]:
```
str
```

16진수

In [65]:
```
a = 0x19AF
```

In [66]:
```
a
```

Out[66]:
```
6575
```

In [67]:
```
type(a)
```

Out[67]:
```
int
```

In [68]:
```
b = hex(6575)
```

In [69]:
```
b  # 결과는 문자열로 나온다.
```

Out[69]:
```
'0x19af'
```

In [70]:
```
type(b)
```

Out[70]:
```
str
```

논리(boolean)형

In [71]:
```
a = True
```

In [72]:
```
a
```

Out[72]:
```
True
```

In [73]:
```
type(a)
```

Out[73]:
```
bool
```

In [74]:
```
b = False
```

In [75]:
```
b
```

Out[75]:
```
False
```

In [76]:
```
type(b)
```

Out[76]:
```
bool
```

In [77]:
```
a = bool(True)
```

In [78]:
```
a
```

Out[78]:
```
True
```

In [79]:
```
type(a)
```

Out[79]:
```
bool
```

In [80]:
```
b = bool(False)
```

In [81]:
```
b
```

Out[81]:
```
False
```

In [82]:
```
type(b)
```

Out[82]:
```
bool
```

In [83]:
```
a + 1
```

Out[83]:
```
2
```

In [84]:
```
b + 1
```

Out[84]:
```
1
```

True는 숫자로 1이며 False는 숫자로 0임을 알 수 있다.

In [85]:
```
a == 1
```

Out[85]:
```
True
```

In [86]:
```
b == 0
```

Out[86]:
```
True
```

컨테이너 (Container)

순차(Sequence)형

리스트 (list)

리스트는 순서가 있는 `가변 (mutable) 객체` 이다.

```
In [87]: a = [1, 2, 3]
```

```
In [88]: a
Out[88]: [1, 2, 3]
```

```
In [89]: type(a)
Out[89]: list
```

```
In [90]: b = [1, 2, 3, 4, 5]
```

```
In [91]: b
Out[91]: [1, 2, 3, 4, 5]
```

```
In [92]: type(b)
Out[92]: list
```

```
In [93]: c = list([1, 2, 3, 4, 5])
```

```
In [94]: c
Out[94]: [1, 2, 3, 4, 5]
```

```
In [95]: type(c)
Out[95]: list
```

파이썬 내장 함수인 `len`은 컨테이너의 요소 개수를 반환하는 함수이다.

```
In [96]: a = [1, 2, 3]
```

```
In [97]: len(a)
Out[97]: 3
```

```
In [98]: b = [5, 8, 2, 1, 4, 2, 3, 6]
```

```
In [99]: len(b)
Out[99]: 8
```

인덱싱

```
In [100]: a = [1, 2, 3, 4, 5]
```

파이썬에서는 sequence 타입의 순서(index)는 0부터 시작한다.

```
In [101]: a[0]  # a의 0번째 요소인 1을 가져온다.
Out[101]: 1
```

```
In [102]: a[2]
Out[102]: 3
```

인덱스가 음수인 경우, 반대로 시작하며 -1인 경우, 맨 마지막 위치에 있는 요소인 5를 가져온다.

```
In [103]: a[-1]
Out[103]: 5
```

```
In [104]: a[-3]
Out[104]: 3
```

```
In [105]: a[-4]
Out[105]: 2
```

```
In [106]: a[1] = 20  # 1번째 요소를 20으로 재할당한다.
```

```
In [107]: a
Out[107]: [1, 20, 3, 4, 5]
```

슬라이싱 (slicing)

```
In [108]: a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [109]: a[1:4]  # 1번째 요소(2)부터 4번째 전(5)까지 가져온다.
Out[109]: [2, 3, 4]

In [110]: b = [1, 6, 2, 4, 8, 5, 3, 10, 9, 7]

In [111]: b[::2]  # n-1번째 인덱스까지 건너뛴 나머지 요소를 가져온다.
Out[111]: [1, 2, 8, 3, 9]

In [112]: b[::3]
Out[112]: [1, 4, 3, 7]

In [113]: b[::4]
Out[113]: [1, 8, 9]
```

리스트 내장 함수 (메소드)

```
In [114]: a = [1, 2, 3, 4, 5]

In [115]: a.index(2)  # 요소 2가 있는 위치는 1번째이므로 1을 반환한다.
Out[115]: 1

In [116]: a.append(6)  # 맨 마지막 위치에 요소 6을 추가시킨다.

In [117]: a
Out[117]: [1, 2, 3, 4, 5, 6]
```

튜플 (tuple)

리스트와 동일하지만 튜플은 `불변 (immutable) 객체` 라는 차이점이 있다.

```
In [118]: a = (1, 2, 3, 4, 5)

In [119]: a
Out[119]: (1, 2, 3, 4, 5)

In [120]: type(a)
Out[120]: tuple

In [121]: b = tuple([1, 2, 3, 4, 5])

In [122]: b
Out[122]: (1, 2, 3, 4, 5)

In [123]: type(b)
Out[123]: tuple

In [124]: a[0]
Out[124]: 1

In [125]: a[3]
Out[125]: 4
```

튜플은 immutable이므로 재할당이 되지 않는다.

```
In [126]: a[1] = 2
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-126-94c4449d0396> in <module>
----> 1 a[1] = 2

TypeError: 'tuple' object does not support item assignment
```

레인지 (range)

range는 시작부터 n-1까지 순서대로 요소를 만들어준다.

```
In [127]: range(1, 10)
Out[127]: range(1, 10)
```

Out[127]: range(1, 10)

In [128]:
```
list(range(1, 10))
```
Out[128]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

In [129]:
```
tuple(range(1, 10))
```
Out[129]: (1, 2, 3, 4, 5, 6, 7, 8, 9)

In [130]:
```
a = range(1, 100)
```

In [131]:
```
a[10]
```
Out[131]: 11

In [132]:
```
a[2:5]
```
Out[132]: range(3, 6)

range 역시 immutable이므로 재할당은 할 수 없다.

In [133]:
```
a[12] = 29
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-133-3ad651257ed2> in <module>
----> 1 a[12] = 29

TypeError: 'range' object does not support item assignment
```

순서 없는(Non-Sequence) 형

집합 (set)

set은 순서가 없으며, `가변 (mutable)` 형이며, `값이 중복되지 않는다.`

In [134]:
```
a = {1, 2, 3, 4, 5}
```

In [135]:
```
a
```
Out[135]: {1, 2, 3, 4, 5}

In [136]:
```
type(a)
```
Out[136]: set

In [137]:
```
b = set([1, 1, 4, 2, 2, 6, 6, 6, 6, 7, 8, 10, 4, 2, 8, 6, 3])
```

In [138]:
```
b
```
Out[138]: {1, 2, 3, 4, 6, 7, 8, 10}

In [139]:
```
type(b)
```
Out[139]: set

`집합` 이라는 명칭에 걸맞게 아래와 같이 합집합, 차집합, 교집합, 대칭차집합을 표현할 수 있다.

In [140]:
```
a = {1, 4, 7}
b = {1, 3, 6, 8, 10}
```

In [141]:
```
a | b  # 합집합
```
Out[141]: {1, 3, 4, 6, 7, 8, 10}

In [142]:
```
a - b  # 차집합
```
Out[142]: {4, 7}

In [143]:
```
a & b  # 교집합
```
Out[143]: {1}

In [144]:
```
a ^ b  # 대칭차집합
```
Out[144]: {3, 4, 6, 7, 8, 10}

딕셔너리 (dict)

dict는 key와 value의 쌍으로 이루어진 자료구조이다.

In [145]:
```
a = {'a': 1, 'b': 2}
```

In [146]:
```
a
```
Out[146]: {'a': 1, 'b': 2}

```
In [147]: b = dict([('two', 2), ('one', 1), ('three', 3)])

In [148]: b
Out[148]: {'two': 2, 'one': 1, 'three': 3}

In [149]: c = dict({'a': 1, 'b': 2})

In [150]: c
Out[150]: {'a': 1, 'b': 2}

In [151]: a['a']
Out[151]: 1

In [152]: a['b']
Out[152]: 2

In [153]: b['two']
Out[153]: 2

In [154]: b['three']
Out[154]: 3
```

Key와 Value를 각각 가져오기

```
In [155]: a = {'a': 1, 'b': 2, 'c': 3}

In [156]: a.keys()  # dict의 모든 key를 가져온다.
Out[156]: dict_keys(['a', 'b', 'c'])

In [157]: a.values()  # dict의 모든 value를 가져온다.
Out[157]: dict_values([1, 2, 3])

In [158]: a.items()  # dict의 key와 value를 쌍으로 가져온다.
Out[158]: dict_items([('a', 1), ('b', 2), ('c', 3)])
```

문자열

문자열은 순서를 가지는 sequence이며 문자열의 종류는 크게 3가지로 분류할 수 있다.

유니코드 문자열 (str)

파이썬 3.x의 문자열은 기본적으로 유니코드(UTF-8) 형식이며 mutable이다.

```
In [159]: a = "hello"

In [160]: a
Out[160]: 'hello'

In [161]: type(a)
Out[161]: str

In [162]: b = "안녕하세요"

In [163]: b
Out[163]: '안녕하세요'

In [164]: type(b)
Out[164]: str
```

파이썬 3.x부터 문자열은 기본적으로 유니코드이므로 유니코드 문자라는 것을 명시하는 `u`는 생략이 가능하다.

```
In [165]: c = u"안녕하세요"

In [166]: c
Out[166]: '안녕하세요'
```

row 포맷 문자열은 `r`을 붙여주면 표현이 가능하다.

```
In [167]: d = "hello\nworld!"

In [168]: d
```

```
Out[168]: 'hello\nworld!'

In [169]: print(d)
          hello
          world!

In [170]: e = r"hello\nworld!"

In [171]: e
Out[171]: 'hello\\nworld!'

In [172]: print(e)
          hello\nworld!
```

문자열을 표현할 때 홑따옴표(`'`)와 쌍따옴표(`"`) 모두 사용이 가능하며 3번 쓰면 독스트링(doc-string) 이라 한다.

```
In [173]: '안녕하세요'
Out[173]: '안녕하세요'

In [174]: "안녕하세요"
Out[174]: '안녕하세요'

In [175]: '''안녕하세요'''
Out[175]: '안녕하세요'

In [176]: """안녕하세요"""
Out[176]: '안녕하세요'
```

앞에 `f` 를 붙여 format string을 사용하면 파이썬 코드의 실행 결과를 문자열에 대입할 수 있다.

```
In [177]: a = f'{sys.maxsize}'

In [178]: type(a)
Out[178]: str

In [179]: a
Out[179]: '9223372036854775807'

In [180]: b = f'{1 == 2}'

In [181]: type(b)
Out[181]: str

In [182]: b
Out[182]: 'False'
```

식별자 역시 유니코드로 선언이 가능하다.

```
In [183]: 일 = 1

In [184]: 일
Out[184]: 1

In [185]: type(일)
Out[185]: int

In [186]: 안녕 = 'hello'

In [187]: 안녕
Out[187]: 'hello'

In [188]: type(안녕)
Out[188]: str
```

아스키(ASCII) 문자열 (bytes)

bytes는 아스키 문자열을 연속적으로 나열한 immutable 시퀀스 타입이다. 즉, bytes는 각 요소 하나가 1byte 크기를 갖는다.

```
In [189]: a = b'abcde'

In [190]: a
Out[190]: b'abcde'

In [191]: type(a)
```

```
Out[191]: bytes
```

```
In [192]: b = b'안녕하세요'  # 유니코드 문자열은 불가능하다.
```

```
  File "<ipython-input-192-201a570e7a44>", line 1
    b = b'안녕하세요'  # 유니코드 문자열은 불가능하다.
        ^
SyntaxError: bytes can only contain ASCII literal characters.
```

```
In [193]: b = b'abcde'
```

bytes는 immutable이므로 재할당이 불가능하다.

```
In [194]: b[0] = 65  # 65는 ASCII 문자로 A를 의미한다.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-194-562c8c0186a3> in <module>
----> 1 b[0] = 65  # 65는 ASCII 문자로 A를 의미한다.

TypeError: 'bytes' object does not support item assignment
```

아스키(ASCII) 문자열 (bytearray)

bytearray도 1바이트 단위의 값을 연속적으로 저장하는 시퀀스 자료형인데, bytes와 차이점은 요소를 변경할 수 있느냐의 차이 즉, mutable이라는 차이가 있다.

```
In [195]: a = bytearray(b'abcde')
```

```
In [196]: a
Out[196]: bytearray(b'abcde')
```

```
In [197]: a[0] = 65
```

```
In [198]: a
Out[198]: bytearray(b'Abcde')
```