IPA 주관 인공지능센터 기본(fundamental) 과정

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

Higher-order Function Part.2

class를 선언하고 instance 객체의 type을 출력한다.

In [1]:
```
class X:
    pass
```

In [2]:
```
a = X()
type(a)
```

Out[2]:
```
__main__.X
```

type

type을 통해 instance 객체를 생성한다.

이에대한 자세한 설명은 meta-class에 대해 다루게 될 때 알 수 있을 것이다.

In [3]:
```
b = type(a)()
type(b)
```

Out[3]:
```
__main__.X
```

In [4]:
```
a = 3
b = type(a)(4)
c = float(b)

print('a:', a, ', type:', type(a))
print('b:', b, ', type:', type(b))
print('c:', c, ', type:', type(c))
```

```
a: 3 , type: <class 'int'>
b: 4 , type: <class 'int'>
c: 4.0 , type: <class 'float'>
```

type을 통해 새로운 type을 정의하는 것도 가능하다.

type을 통해 int 타입을 정의한다.

In [5]:
```
x = type('int', (), {})
x
```

Out[5]:
```
__main__.int
```

type이 어떤식으로 정의되었는지 확인해본다.

In [6]:
```
help(type)
```

```
Help on class type in module builtins:

class type(object)
 |  type(object_or_name, bases, dict)
 |  type(object) -> the object's type
 |  type(name, bases, dict) -> a new type
 |
 |  Methods defined here:
 |
 |  __call__(self, /, *args, **kwargs)
 |      Call self as a function.
 |
 |  __delattr__(self, name, /)
 |      Implement delattr(self, name).
 |
 |  __dir__(...)
 |      __dir__() -> list
 |      specialized __dir__ implementation for types
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __instancecheck__(...)
 |      __instancecheck__() -> bool
 |      check if an object is an instance
 |
 |  __new__(*args, **kwargs)
 |      Create and return a new object.  See help(type) for accurate signature.
 |
```

```
|  __prepare__(...)
|      __prepare__() -> dict
|      used to create the namespace for the class statement
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __setattr__(self, name, value, /)
|      Implement setattr(self, name, value).
|
|  __sizeof__(...)
|      __sizeof__() -> int
|      return memory consumption of the type object
|
|  __subclasscheck__(...)
|      __subclasscheck__() -> bool
|      check if a class is a subclass
|
|  __subclasses__(...)
|      __subclasses__() -> list of immediate subclasses
|
|  mro(...)
|      mro() -> list
|      return a type's method resolution order
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  __abstractmethods__
|
|  __dict__
|
|  __text_signature__
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __base__ = <class 'object'>
|      The most base type
|
|  __bases__ = (<class 'object'>,)
|
|  __basicsize__ = 864
|
|  __dictoffset__ = 264
|
|  __flags__ = 2148291584
|
|  __itemsize__ = 40
|
|  __mro__ = (<class 'type'>, <class 'object'>)
|
|  __weakrefoffset__ = 368
```

iter(Iterable)

- iter()는 iter(Iterable)와 같이 사용하며 그 Iterable 객체의 iterator를 반환한다.

In [7]:
```
i = iter([
    1,
    2,
    3,
    4,
])
```

In [8]:
```
dir(i)
```

Out[8]:
```
['__class__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__length_hint__',
 '__lt__',
 '__ne__',
 '__new__',
 '__next__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__']
```

In [9]:
```
print(i.__next__())  # 1번째 요소를 가져온다.
print(i.__next__())  # 2번째 요소를 가져온다.
print(i.__next__())  # 3번째 요소를 가져온다.
print(i.__next__())  # 4번째 요소를 가져온다.
print(i.__next__())  # 더 이상 요소가 존재하지 않으므로 StopIteration 예외를 발생시킨다.
```

```
1
2
3
4

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-9-b71f289f2e66> in <module>
      3 print(i.__next__()) # 3번째 요소를 가져온다.
      4 print(i.__next__()) # 4번째 요소를 가져온다.
----> 5 print(i.__next__()) # 더 이상 요소가 존재하지 않으므로 StopIteration 예외를 발생시킨다.

StopIteration:
```

In [10]:
```
class CamelName:
    a = 10

    def yy(self):
        return self.a
```

In [11]:
```
a = CamelName()
a.a
```

Out[11]: 10

class에 정의되어 있지 않은 새로운 instance attribute를 선언할 수도 있다.

In [12]:
```
a.b = 10
a.b
```

Out[12]: 10

vars를 통해 instance attribute를 확인해본다.

In [13]:
```
vars(a)
```

Out[13]: {'b': 10}

기존에 python은 method를 호출할 때 아래와 같이 사용했다.

In [14]:
```
CamelName.yy(a)
```

Out[14]: 10

syntax sugar

In [15]:
```
a.yy()  # 위 문법의 간략한 표현이다.
```

Out[15]: 10

In [16]:
```
def fibonacci():
    a, b = 1, 1
    while True:
        yield a
        a, b = b, a + b
```

yield를 반환하는 함수이므로 generator가 생성된다.

In [17]:
```
fibo = fibonacci()
type(fibo)
```

Out[17]: generator

next로 generator를 호출할 때마다 새로운 객체를 반환시킨다.

In [18]:
```
next(fibo)
```

Out[18]: 1

itertools 살펴보기

In [19]:
```
from itertools import tee, accumulate
```

tee(iterator, n)

- iterator 객체를 n만큼 복사하여 독립된 객체들을 생성해줍니다.

In [20]:
```
tee([
    1,
    2,
    3,
    4,
], 3)
```

```
Out[20]: (<itertools._tee at 0x7fd7a03bf3c8>,
          <itertools._tee at 0x7fd7900e2348>,
          <itertools._tee at 0x7fd79006e188>)
```

accumulate

```
In [21]: ac = accumulate([
             1,
             2,
             3,
             4,
         ])
```

```
In [22]: next(ac)
Out[22]: 1
```

```
In [23]: next(ac)
Out[23]: 3
```

```
In [24]: next(ac)
Out[24]: 6
```

```
In [25]: next(ac)
Out[25]: 10
```

Fibonacci

```
In [26]: s, t = tee(fibonacci())
         pairs = zip(t, accumulate(s))
         for _, (fib, total) in zip(range(7), pairs):
             print(fib, total)
```

```
1 1
1 2
2 4
3 7
5 12
8 20
13 33
```

map vs comprehension 속도 비교

- 아래의 결과를 보면 comprehension이 속도가 빠르다는 것을 확인할 수 있다.

```
In [27]: %timeit list(map(lambda x: x+1, [1, 2, 3, 4]))
406 ns ± 0.334 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

```
In [28]: %timeit[x+1 for x in [1, 2, 3, 4]]
170 ns ± 2.02 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

range

```
In [29]: %timeit sum(range(10000))
94.7 µs ± 87.1 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

아래는 식이 아니므로 불가능하다.

```
In [30]: y = x for x in range(10000)
  File "<ipython-input-30-16ccf922e70f>", line 1
    y = x for x in range(10000)
          ^
SyntaxError: invalid syntax
```

아래와 같이 함수 인자로 넣어주면 python 내부에서 한번 더 체크하여 가능하도록 바꿔준다.

```
In [31]: %timeit sum(x for x in range(10000))
290 µs ± 5.04 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

comprehansion

```
In [32]: %timeit sum([x for x in range(10000)])
313 µs ± 2.42 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```