파이썬 입문

한국폴리텍대학

2023.04

일급 객체, 퍼스트 클래스 오브젝트 (first-class object, first-class citizen)

first-class 정의

- 런타임에 생성 가능
- 변수나 데이터 구조체의 요소에 할당 가능
- 함수 인수(argument)로 전달 가능
- 함수 결과로 리턴 가능

first-class function

First-class 기능을 가진 함수

```
def square(x1):
        return x1 * x1

s = square          #  변수에 할당
s(5)                #  25
```

```
def add(x1, x2):
        return x1 + x2

def sub(x1, x2):
        return x1 - x2

def mul(x1, x2):
        return x1 * x2

def div(x1, x2):
        return x1 / x2

data = [add, sub, mul, div]   # 요소에 할당

print(data[0](10, 5))    # 15
print(data[1](10, 5))    # 5
print(data[2](10, 5))    # 50
print(data[3](10, 5))    # 2.0
```

일급 객체, 퍼스트 클래스 오브젝트 (first-class object, first-class citizen)

first-class function

```
def square(x1):
    return x1 * x1

def power(func, n):
    return func(n)

power(square, 5)     #  함수 인수로 전달
                     #  25
```

```
def calc():
    def add(x1, x2):
        y = x1 + x2
        return y
    return add

c = calc                #  함수 결과로 리턴
print(c(8, 3)           #  11
```

일급 객체, 퍼스트 클래스 오브젝트 (first-class object, first-class citizen)

```
class calc1():
        def add(self, x1, x2):
                return x1 + x2

class calc2():
        def sub(self, x1, x2):
                return x1 - x2

c1 = calc1()
c2 = calc1()
c3 = calc2()
c4 = calc2()

data = [c1, c2, c3, c4]   # list 요소에 할당
#data = (c1, c2, c3, c4)   # tuple 요소에 할당

print(data[0].add(10, 5))    # 25
print(data[1].add(10, 5))    # 15
print(data[2].sub(10, 5))    # 15
print(data[3].sub(10, 5))    # 5
```

```
class calc1():
        def add(self, x1, x2):
                return x1 + x2

class calc2():
        def sub(self, x1, x2):
                return x1 - x2

c1 = calc1()
c2 = calc1()
c3 = calc2()
c4 = calc2()

data = {'1': c1, '2': c2, '3': c3, '4': c4}   # dict 요소에 할당
#data = {c1, c2, c3, c4}    # set은 순서가 없음
                            # indexing, slicing 기능 없음

print(data['1'].add(10, 5))    # 25
print(data['3'].sub(10, 5))    # 15
```

callable

callable_object_name()

1. 호출 가능한 클래스 인스턴스
2. 메소드
3. 함수

등의 객체를 의미

```
def callfunc():
        print("callable func")

class callable_unit():
        def __call__(self):
                print("callable unit")
                return self

class unit():
        def __init__(self):
                print("callable unit")
```

```
callfunc()                      # callable func
print(callable(callfunc))       # True

c1 = callable_unit()
c1()                            # callable unit
print(callable(c1)              # True

c2 = unit()
c2()                            # TypeError
                                # object is not callable
print(callable(c2)              # False
```

객체 슬라이싱 __getitem__

리스트, 튜플, 문자열 등은 []를 이용해
인덱싱, 슬라이싱이 가능함.

__getitem__은 클래스의 인스턴스도 인덱싱,
슬라이싱이 가능하게 함.

subscriptable: 인덱싱(indexing), 슬라이싱(slicing)이 가능한.
예) list, dict, tuple, str 등

```
class Customlist:
        def __init__(self, *users):
                self.users = list(users)

        def __getitem__(self, index):
                return self.users[index]

customlist = Customlist(1, 2, 3, 4, 5, 6, 7)
print(customlist.users[5])      # 6
print(customlist[3])            # 4
```

*args : arguments --> 임의의 개수 인자 받기

**kwargs : keyword arguments --> 임의의 개수 dict 형식의 인수 받기

쓰레드 thread

```
import threading
import time

def sum(x1, x2, x3):
        total = x1
        count = x3
        while(count > 0):
                time.sleep(2)
                total += x2
                print("sum: ", total)
                count -= 1

t = threading.Thread(target=sum, args=(0, 5, 10), daemon=False))
t.start()

print("main thread start")
print("main thread end")
```

쓰레드: 작업의 단위

program, process, thread (application)

--> daemon=False가 기본값

쓰레드 데몬 thread daemon

```
import threading
import time

def sum(x1, x2, x3):
        total, count = x1, x3
        while(count > 0):
                time.sleep(2)
                total += x2
                print("₩nsub thread sum: {}".format(total))
                count -= 1

t = threading.Thread(target=sum, args=(0, 5, 10), daemon=True))    --> daemon=False가 기본값
#t.daemon = True
t.start()
#t.join()

print("main thread start")
i = 10
while i > 0:
        print("₩nmain thread working")
        time.sleep(1)
        i -= 1
print("main thread end")
```

쓰레드: 작업의 단위

program, process, thread (application)

```
# daemon: False --> main thread 종료 후에도 sub thread 계속 수행
# daemon: True --> main thread 종료되면 sub thread 종료
# join(): --> sub thread 종료 후 다음 line 수행
```