Előadás -6

Programozás Pythonban

Instructor : Dr. AALWAHAB DHULFIQAR Advisor : Dr. Tejfel Mate

Mit fogsz tanulni:

- Öröklés
- Hogyan találja meg a Python a tulajdonságokat és metódusokat
- A gyémánt probléma
- Iterátor
- A yield utasítás
- Listák megértése
- A lambda függvény
- Fájlok elérése Python kódból

Öröklés

```
class Star:
    def__init__(self, name, galaxy):
        self.name = name
        self.galaxy = galaxy

sun = Star("Sun", "Milky Way")
print(sun)
```

<__main__.Star object at 0x7f8bb6eff350>

```
class school:
    def__init__(self, name, age):
        self.name = name
        self.age = age
    def__str__(self):
        return self.name + ' is ' + self.age
sun = school("Alex", '13')
print(sun)
```

Alex in 13

```
class Vehicle:
    pass
class LandVehicle(Vehicle):
    pass
class TrackedVehicle(LandVehicle):
    pass
my_vehicle = Vehicle()
my_land_vehicle = LandVehicle()
my_tracked_vehicle = TrackedVehicle()

for obj in [my_vehicle, my_land_vehicle, my_tracked_vehicle]:
    for cls in [Vehicle, LandVehicle, TrackedVehicle]:
        print(isinstance(obj, cls), end="\t")
    print()
```

```
class SampleClass:
    def_init_(self, val):
        self.val = val


object_1 = SampleClass(0)
object_2 = SampleClass(2)
object_3 = object_1
object_3.val += 1

print(object_1 is object_2)
print(object_2 is object_3)
print(object_3 is object_1)
print(object_1.val, object_2.val, object_3.val)

string_1 = "Mary had a little "
string_2 = "Mary had a little lamb"
string_1 += "lamb"

print(string_1 == string_2, string_1 is string_2)
```

False

False

True

1 2 1

True False

Hogyan találja meg a Python a tulajdonságokat és módszereket

```
class Super:
    def init (self, name):
        self.name = name

    def str (self):
        return "My name is " + self.name + "."


class Sub(Super):
    def init (self, name):
    Super. init (self, name)


obj = Sub("Andy")

print(obj)
```

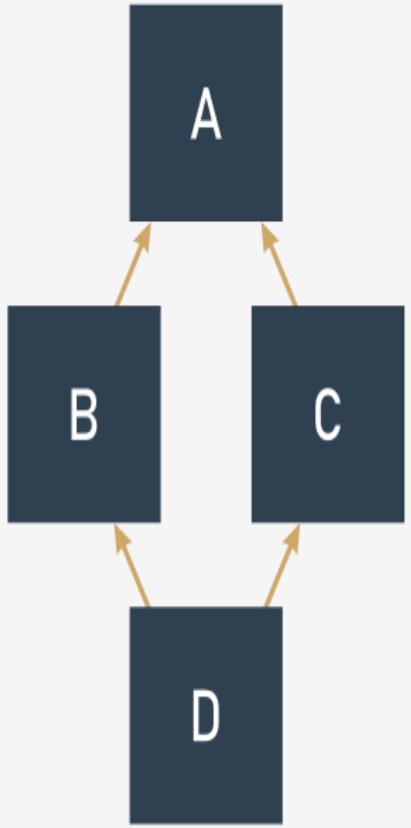
```
class Super:
    def init (self, name):
        self.name = name

    def str (self):
        return "My name is " + self.name + "."


class Sub(Super):
    def init (self, name):
        super(). init (name)


obj = Sub("Andy")

print(obj)
```

```
class Super:
    supVar = 1


class Sub(Super):
    subVar = 2


obj = Sub()

print(obj.subVar)
print(obj.supVar)
```

```
class Left:
    var = "L"
    var_left = "LL"
    def fun(self):
        return "Left"


class Right:
    var = "R"
    var_right = "RR"
    def fun(self):
        return "Right"


class Sub(Left, Right):
    pass


obj = Sub()

print(obj.var, obj.var_left,
obj.var_right, obj.fun())
```

```
import time

class Tracks:
    def change_direction(self, left, on):
        print("tracks: ", left, on)


class Wheels:
    def change_direction(self, left, on):
        print("wheels: ", left, on)


class Vehicle:
    def __init__(self, controller):
        self.controller = controller

    def turn(self, left):
        self.controller.change_direction(left,
True)
        time.sleep(0.25)
        self.controller.change_direction(left,
False)


wheeled = Vehicle(Wheels())
tracked = Vehicle(Tracks())

wheeled.turn(True)
tracked.turn(False)
```

A gyémánt probléma




```
class Top:
    def m_top(self):
        print("top")


class Middle_Left(Top):
    def m_middle(self):
        print("middle_left")


class Middle_Right(Top):
    def m_middle(self):
        print("middle_right")


class Bottom(Middle_Left, Middle_Right):
    def m_bottom(self):
        print("bottom")


object = Bottom()
object.m_bottom()
object.m_middle()
object.m_top()
```

Generators

```
for i in range(5):
    print(i)
```

Iterator

An iterator must provide two methods:

1. __iter__() which should return the object itself and which is invoked once (it's needed for Python to successfully start the iteration)
2. __next__() which is intended to return the next value (first, second, and so on) of the desired series - it will be invoked by the for/in statements in order to pass through the next iteration; if there are no more values to provide, the method should raise the StopIteration exception.

Iterator

```
class Fib:
    def __init__(self, nn):
        print("__init__")
        self.__n = nn
        self.__i = 0
        self.__p1 = self.__p2 = 1

    def __iter__(self):
        print("__iter__")
        return self

    def __next__(self):
        print("__next__")
        self.__i += 1
        if self.__i > self.__n:
            raise StopIteration
        if self.__i in [1, 2]:
            return 1
        ret = self.__p1 + self.__p2
        self.__p1, self.__p2 = self.__p2, ret
        return ret


for i in Fib(10):
    print(i)
```

A yield utasítás

```
def fun(n):
    for i in range(n):
        return i
```

```
def fun(n):
    for i in range(n):
        yield i
```

```
def fun(n):
    for i in range(n):
        yield I

for v in fun(5):
    print(v)
```

```
def powers_of_2(n):
    power = 1
    for i in range(n):
        yield power
        power *= 2

for v in powers_of_2(8):
    print(v)
```

Hogyan építsünk saját generátort

Listák comprehensions

```
def powers_of_2(n):
    power = 1
    for i in range(n):
        yield power
        power *= 2



print(t)
```

listaértelmezés

The `list()` függvény

```
def powers_of_2(n):
    power = 1
    for i in range(n):
        yield power
        power *= 2



print(t)
```

The in operator

```
def powers_of_2(n):
    power = 1
    for i in range(n):
        yield power
        power *= 2
for i in range(20):
    if i in powers_of_2(4):
        print(i)
```

```
def fibonacci(n):
    p = pp = 1
    for i in range(n):
        if i in [0, 1]:
            yield 1
        else:
            n = p + pp
            pp, p = p, n
            yield n

fibs = list(fibonacci(10))
print(fibs)
```

Tekintse meg az iterátor példát a előző dián!

A lambda függvény

A lambda függvény egy név nélküli függvény (nevezhetjük névtelen függvénynek is).

lambda paraméterek: kifejezés

```
two = lambda: 2
sqr = lambda x: x * x
pwr = lambda x, y: x ** y

for a in range(-2, 3):
    print(sqr(a), end=" ")
    print(pwr(a, two()))
```

```
4 4
1 1
0 0
1 1
4 4
```

1. Az első lambda egy névtelen paraméter nélküli függvény, amely mindig 2-t ad vissza, mivel hozzárendeltük egy two nevű változóhoz, mondhatjuk, hogy a függvény már nem névtelen, és a nevet használhatjuk a meghívásához.
2. A második egy egy paraméteres anonim függvény, amely a négyzetre vetített argumentum értékét adja vissza. Ezt is így neveztük el.
3. A harmadik lambda két paramétert vesz fel, és az első paraméter értékét adja vissza a második paraméter hatványára emelve. A lambdát hordozó változó neve magáért beszél. Nem használjuk a pow-t, hogy elkerüljük az összetévesztést az azonos nevű és azonos célú beépített függvénnyel.

Hogyan használjuk a lambdákat és mire?

```
def print_function(args, fun):
    for x in args:
        print('f(', x,')=', fun(x), sep='')

def poly(x):
    return 2 * x**2 - 4 * x + 2

print_function([x for x in range(-2, 3)], poly)
```

```
f(-2)=18
f(-1)=8
f(0)=2
f(1)=0
f(2)=2
```

```
def print_function(args, fun):
    for x in args:
        print('f(', x,')=', fun(x), sep='')

print_function([x for x in range(-2, 3)], lambda x: 2 * x**2 - 4 * x + 2)
```

Lambdák és a map() függvény

map(function, list)

a második map() argumentum lehet bármilyen iterálható entitás (pl. egy tuple, vagy csak egy generátor).

```
list_1 = [x for x in range(5)]
list_2 = list(map(lambda x: 2 ** x, list_1))
print(list_2)

for x in map(lambda x: x * x, list_2):
    print(x, end=' ')
print()
```

Lambdák és a filter() függvény

megszűri a második argumentumát, miközben az első argumentumként megadott függvényből származó irányok vezérlik.

```
from random import seed, randint

seed()
data = [randint(-10,10) for x in range(5)]
filtered = list(filter(lambda x: x > 0 and x % 2 == 0, data))

print(data)
print(filtered)
```

A brief look at closures
A bezárások rövid áttekintése

```
def outer(par):
    loc = par

    def inner():
        return loc
    return inner

var = 1
fun = outer(var)
print(fun())
```

1

Az outer() meghívása során visszaadott függvény egy lezárás.

```
def make_closure(par):
    loc = par
    def power(p):
        return p ** loc
    return power
fsqr = make_closure(2)
fcub = make_closure(3)
for i in range(5):
    print(i, fsqr(i), fcub(i))
```

0 0 0

1 1 1

2 4 8

3 9 27

4 16 64

Fájlok elérése Python kódból

Windows

```
C:\directory\file
```

Linux

```
/directory/files
```

Minden Pythonban írt program (és nem csak Pythonban, mert ez a konvenció gyakorlatilag minden programozási nyelvre vonatkozik) nem közvetlenül kommunikál a fájlokkal, hanem valamilyen absztrakt entitásokon keresztül, amelyeket különböző nyelveken vagy környezetekben másképp neveznek - a leggyakrabban használt kifejezések a handles vagy a stream.

A programozó többé-kevésbé gazdag függvénykészlettel/módszerekkel rendelkezve képes bizonyos műveleteket végrehajtani a folyamon, amelyek az operációs rendszer kernelében található mechanizmusok segítségével befolyásolják a valós fájlokat.

Az adatfolyamon (streamon) két alapvető művelet végezhető:

- olvasás a folyamból: az adatrészletek a fájlból kerülnek elő, és a program által kezelt memóriaterületre (pl. egy változóra) kerülnek;
- írás a folyamba: a memóriából származó adatrészletek (pl. egy változó) átkerülnek a fájlba.

Opening the streams A stream megnyitása

```
stream = open(file, mode = 'r', encoding = None)
```

Text mode	Binary mode	Description
`rt`	`rb`	read
`wt`	`wb`	write
`at`	`ab`	append
`r+t`	`r+b`	read and update
`w+t`	`w+b`	write and update

```
try:
    stream = open("C:\Users\User\Desktop\file.txt", "rt")
    # Processing goes here.
    stream.close()
except Exception as exc:
    print("Cannot open the file:", exc)
```

sys.stdin

stdin (as standard input)
the stdin stream is normally associated with the keyboard, pre-open for reading and regarded as the primary data source for the running programs;
the well-known input() function reads data from stdin by default.

sys.stdout

stdout (as standard output)
the stdout stream is normally associated with the screen, pre-open for writing, regarded as the primary target for outputting data by the running program;
the well-known print() function outputs the data to the stdout stream.

sys.stderr

stderr (as standard error output)
the stderr stream is normally associated with the screen, pre-open for writing, regarded as the primary place where the running program should send information on the errors encountered during its work;

Closing streams

A stream utolsó művelete a lezárás.

```
stream.close()
```

stream problémák diagnózisa

```
try:
    # Some stream operations.
except IOError as exc:
    print(exc.errno)
```

```
from os import strerror

try:
    s = open("c:/users/user/Desktop/file.txt", "rt")
    # Actual processing goes here.
    s.close()
except Exception as exc:
    print("The file could not be opened:",
strerror(exc.errno))
```

constants useful for detecting stream errors:

errno.EACCES → Permission denied
errno.EBADF → Bad file number
errno.EEXIST → File exists
errno.EFBIG → File too large
errno.EISDIR → Is a directory
errno.EMFILE → Too many open files
errno.ENOENT → No such file or directory
errno.ENOSPC → No space left on device

```
try:
    s = open("c:/users/user/Desktop/file.txt", "rt")
    # Actual processing goes here.
    s.close()
except Exception as exc:
    if exc.errno == errno.ENOENT:
        print("The file doesn't exist.")
    elif exc.errno == errno.EMFILE:
        print("You've opened too many files.")
```

Szöveges fájl feldolgozása

```
# # A tzop.txt megnyitása olvasási üzemmódban,
fájlobjektumként való visszaadása:

stream = open("read.txt", "rt", encoding = "utf-8")

print(stream.read()) # printing the content of the file
```

Szöveges fájlok feldolgozása:
readlines()

```
from os import strerror

try:
    cnt = 0
    s = open('text.txt', "rt")
    ch = s.read(1)
    while ch != '':
        print(ch, end='')
        cnt += 1
        ch = s.read(1)
    s.close()
    print("\n\nCharacters in file:", cnt)
except IOError as e:
    print("I/O error occurred: ", strerror(e.errno))
```

```
from os import strerror

try:
    cnt = 0
    s = open('text.txt', "rt")
    content = s.read()
    for ch in content:
        print(ch, end='')
        cnt += 1
    s.close()
    print("\n\nCharacters in file:", cnt)
except IOError as e:
    print("I/O error occurred: ", strerr(e.errno))
```

szöveges fájlok kezelése: `write()`

```
from os import strerror

try:
        fo = open('newtext.txt', 'wt') # A new file (newtext.txt)
is created.
        for i in range(10):
                s = "line #" + str(i+1) + "\n"
                for ch in s:
                        fo.write(ch)
        fo.close()
except IOError as e:
        print("I/O error occurred: ", strerror(e.errno))
```

```
from os import strerror

try:
    fo = open('newtext.txt', 'wt')
    for i in range(10):
        fo.write("line #" + str(i+1) + "\n")
    fo.close()
except IOError as e:
    print("I/O error occurred: ", strerror(e.errno))
```

Bytearrays

```
data = bytearray(10)
print (data)

for i in range(len(data)):
    data[i] = 10 - i

for b in data:
    print(hex(b))
```

Hogyan írjunk byte-okat egy streamből

```
from os import strerror

data = bytearray(10)

for i in range(len(data)):
    data[i] = 10 + i

try:
    bf = open('file.bin', 'wb')
    bf.write(data)
    bf.close()
except IOError as e:
    print("I/O error occurred:", strerror(e.errno))

# Your code that reads bytes from the stream should go here.
```

Hogyan olvassunk byte-okat egy streamből

```
from os import strerror

try:
    bf = open('file.bin', 'rb')
    data = bytearray(bf.read())
    bf.close()

    for b in data:
        print(hex(b), end=' ')

except IOError as e:
    print("I/O error occurred:",
strerror(e.errno))
```

Találkozunk a laborban ☺