**lambda :**

در پایتون ، یک تابع ناشناس ممکن است تابعی باشد که بدون شهرت تعریف شده باشد. هنگام تعریف توابع عادی، ما از کلمه کلیدی def در پایتون استفاده می کنیم، اما هنگام تعریف توابع ناشناس از کلمه کلیدی lambda استفاده می کنیم. لامبدا روشی سریع برای ایجاد تابع است. عبارات لامبدا باید به ساده ترین شکل ممکن و به دور از هرگونه پیچیدگی نوشته شوند

## **# چطور از lambda استفاده کنیم؟**

ساختار نگارش کلی عبارات لامبدا به شکل زیر است:

lambda (arguments): manipulation(arguments)

ابتدا با استفاده از کلمه کلیدی lambda مشخص میکنید که قراره یک لامبدا داشته باشید. بعد از اون آرگومان هایی که قرار است عبارت لامبدا داشته باشه رو مینویسید. و در آخر کاری قراره روی آرگومان ها انجام بشه رو مینویسید. دقت کنید که عبارت lambda هر تعدادی که بخواهید میتواند آرگومان داشته باشد اما فقط یک manipulation خواهد داشت.

### **+ یک مثال از عبارت لامبدا**

به مثال زیر دقت کنید:

add = lambda x, y: x + y

print( add(4, 6) )

در مثال بالا یک عبارت لامبدا ساختیم که دو مقدار x و y رو میگیره و اونها رو با هم جمع میکنه. و در آخر لامبدا رو داخل یک متغیر ذخیره میکنیم تا بتونیم بعدا ازش استفاده کنیم. در خط آخر هم لامبدا رو صدا زدیم و دو عدد رو بهش فرستادیم. اگه کد بالا رو اجرا کنید مقدار ۱۰ رو به شما برمیگردونه. دقت کنید که لامبدا به صورت اتوماتیک مقدار نهایی رو return میکنه.

+ چرا از عبارت لامبدا استفاده کنیم؟

همانطور که در قسمت قبل مشاهده کردید، پایتون با لامبدا به شکل یکسان با فانکشن‌های معمولی برخورد میکند. به نوعی، lambda شکلی جمع و جور برای نوشتن توابع ارائه می دهد که یک عبارت را باز می گرداند. با این حال، شما باید بدانید که استفاده از لامبدا چه زمانی مناسب است و چه زمانی از آنها اجتناب کنید.

یکی از مهمترین موارد استفاده رایج برای lambda، برنامه نویسی فانکشنال است زیرا پایتون از پارادایم (یا سبک) برنامه نویسی که به عنوان برنامه نویسی فانکشنال شناخته می شود پشتیبانی می کند.

لامبدا به شما این امکان را می دهد که یک تابع را به عنوان پارامتر به یک تابع دیگر (به عنوان مثال، در map، filter و غیره) ارائه دهید. در چنین مواردی، استفاده از lambda به لطف ایجاد یک تابع یک بارمصرف، مفید است.

در محیط production، هرگز نباید توابع lambda پیچیده را بنویسید، زیرا رمزگشایی برای کد نویسان که کد شما را حفظ می کنند بسیار دشوار است. اگر خودتان را در حال ساختن عبارات پیچیده تک خطی کشف کردید، بهتر است با def یک فانکشن کامل بنویسید.

## **# استفاده از lambda با filter**

تابع filter برای فیلتر کردن برخی عناصر خاص از یک دنباله استفاده می شود. دنباله مورد استفاده در این تابع یک [آبجکت ‌iterable](https://www.mongard.ir/one_part/53/creating-iterable-objects-python/) مانند لیست ها ، مجموعه ها ، چندتایی ها و غیره است.

sequences = [10,2,8,7,5,4,3,11,0, 1]

filtered\_answer = filter (lambda x: x > 6, sequences)

print(list(filtered\_answer))

# OUTPUT

# [10, 8, 7, 11]

# استفاده از lambda با map

تابع map برای استفاده از یک عملیات خاص برای هر عنصر در یک دنباله استفاده می شود.

sequences = [10,2,8,7,5,4,11]

squared\_result = map (lambda x: x\*x, sequences)

print(list(squared\_result))

# OUTPUT

# [100, 4, 64, 49, 25, 16, 121]

=================================================

**\_\_str\_\_ vs \_\_repr\_\_ :**

این دو متد برای زمانی هستش که شما میخواید یک آبجکت رو چاپ کنید:

class Person:

def \_\_init\_\_(self, name):

self.name = name

p1 = Person('amir')

print(p1)

زمانی که کد بالا رو اجرا میکنیم به شکل زیر نتیجه رو به ما نشون میده:

<\_\_main\_\_.Person object at 0x7f2772b0cf50>

پاسخی که پایتون برای ما آورده آنچنان مطلوب نیستش و نمیشه چیز زیادی ازش فهمید. برای تغییر دادن این رفتار پایتون از متد str استفاده میکنیم:

class Person:

def \_\_init\_\_(self, name):

self.name = name

def \_\_str\_\_(self):

return self.name

p1 = Person('amir')

print(p1)

خب حالا اگه کد بالا رو اجرا کنید نتیحه بهتری رو نمایش میده. در کد بالا به پایتون گفتیم زمانی که یک آبجکت رو چاپ کردیم مقدار name رو نشون بده:

amir

حالا متد repr چیه؟ متد repr هم دقیقا زمانی استفاده میشه که شما یک آبجکت رو صدا بزنید. اما تفاوتی که داره اینکه اگه کدتون رو از طریق ترمینال اجرا کنید متد repr فعال خواهد شد. درواقع متد repr برای زمانی هستش که شما قراره یک نتیجه رو به برنامه نویس دیگه نشون بدید اما متد str برای زمانی هستش که شما میخواید یک نتیجه رو به کاربر نشون بدین.

برای درک این موضوع داخل ترمنیال، با استفاده از دستور python وارد مفسر پایتون بشید.

حالا کد زیر رو وارد میکنیم:

>>> import datetime

>>> now = datetime.datetime.now()

>>> str(now)

'2019-8-15 10:29:34.786394'

>>> repr(now)

datetime.datetime(2019, 8, 15, 10, 29, 34, 786394)

در کد بالا ما زمان الآن رو ریختیم داخل متغیر now.

زمانی که با استفاده از متد str متغیر now رو صدا میزنیم میبینید که نتیجه به شکل یک زمان نشون داده میشه.

اما زمانی که متغیر now رو با متد repr صدا میزنیم نتیجه رو به شکل یک فانکشن میاره.

تفاوت این دو متد اینجا مشخص میشه. متد str برای نشون دادن یک نتیجه به کاربر عادی اما متد repr با هدف دیباگ کردن برنامه و نشون دادن یک پیغام به برنامه نویس دیگه استفاده میشه.

به صورت دیفالت داخل ترمینال متد repr صدا زده می‌شود

=================================================

**shallow and deep copy :**

shallow copy یعنی کپی سطحی و deep copy یعنی کپی عمیق.

نکته ای که همین اول کار باید بدونید اینکه این بحث فقط در مورد آبجکت های mutable مثل list, dict, set وجود داره و آبجکت هایی که immutable هستند مثل tuple ها این بحث درموردشون وجود نداره.

زمانی که شما سعی در کپی گرفتن از یک آبجکت mutable رو دارید اون چیزی که شما انتظار دارید با چیزی که پایتون قراره انجام بده متفاوت هستش. به مثال زیر دقت کنید:

a = [1, 2, 3, 4]

b = a

من در خط اول یک لیست دارم. برای کپی گرفتن از این لیست ساده ترین روشی که به ذهنمون میرسه اینکه اون لیست به یک متغیر دیگه منسوب کنیم، دقیقا مثل خط دوم. اما اینکار یک کپی از لیست نمیگیره و فقط یک اشاره گر جدید به اون لیست ایجاد میکنه.

اگر هم بخواید مطمئن بشید میتونید id های متغیرها رو چاپ کنید:

print( id(a) )

print( id(b) )

کد بالا رو که اجرا کنید میبینید که یک عدد رو نشون میده. به این میگیم shallow copy در پایتون. حالا من سعی میکنیم که لیست b رو تغییر بدم:

a = [1, 2, 3, 4]

b = a

b[0] = 11

print(a)

print(b)

در کد بالا من لیست b رو تغییر دادم اما لیست a هم تغییر خواهد کرد. چرا؟ چون b فقط یک اشاره گر بود به لیست a:

[11, 2, 3, 4]

[11, 2, 3, 4]

برای حل این مشکل ما سه تا متد داریم: list(), dict(), set()

b = list(a)

b[0] = 11

این سه تا متد deep copy میگیرن. حالا اگه لیست b رو تغییر بدین میبینید که لیست a تغییری نکرده.

[1, 2, 3, 4] #a

[11, 2, 3, 4] #b

اما این سه تا متد در زمانی که شما آبجکت های تو در تو دارید به درستی کار نخواهند کرد.

a = [1, 2, 3, 4, [5, 6]]

b = list(a)

b[4][0] = 55

print(a) # [1, 2, 3, 4, [55, 6]]

print(b) # [1, 2, 3, 4, [55, 6]]

همونطور که میبینید زمانی که لیست هامون تو در تو هستند باز هم هردو لیست تغییر میکنن. برای حل کامل مشکل میتونید از ماژول copy استفاده کنید:

import copy

a = [1, 2, 3, 4]

b = copy.copy(a) # shallow copy

c = copy.deepcopy(a) # deep copy

در این ماژول دوتا متد داریم که برامون هم shallow copy و هم deep copy میگیرن.

=================================================

**return None in pyhton :**

در پایتون زمانی که داخل فانکشن هاتون مقداری رو return نکنید، پایتون به صورت اتوماتیک None رو return خواهد کرد. به سه روش این کار انجام میشه.

روش اول زمانی هستش که شما به صورت صریح اعلام کنید که میخواید None رو return کنید:

def show(value):

if value:

return value

else:

return None

در مثال بالا اگه به فانکشن مقداری رو ارسال نکنید اون فانکشن برای شما None رو return خواهد کرد.

در روش دوم شما فقط کلمه return رو مینویسید:

def show(value):

if value:

return value

else:

return

در مثال بالا هم خود پایتون به صورت اتوماتیک مقدار None رو return میکنه.

و روش سوم زمانی هستش که شما بلاک else رو کلا نمینویسید:

def show(value):

if value:

return value

در مثال بالا هم اگه مقداری به فانکشن ارسال نشه پایتون به صورت اتوماتیک مقدار None رو return خواهد کرد.

=================================================

**json in pyhton :**

در این آموزش ، نحوه ذخیره داده ها در پایتون با استفاده از ماژول JSON را بررسی می کنیم. ما همچنین نحوه استفاده از متدهای json.dump () و json.dumps () ، متدهای json.load () و json.loads () و تفاوت آنها را یاد می گیریم. سرانجام، ما به نحوه serialize و deserialize کردن JSON به Object در پایتون خواهیم پرداخت.

## **# چرا داده ها را در پایتون با استفاده از ماژول JSON ذخیره می کنیم؟**

1. توسعه دهندگان را قادر می سازد تا ساختارهای داده ساده را در یک فایل ریخته و در صورت نیاز بارگذاری کنند.

2. با استفاده از JSON می توان داده ها را بین برنامه های پایتون به اشتراک گذاشت.

3. فرمت JSON از پلتفرم یا زبان مستقل است. وقتی داده ها را با فرمت JSON ذخیره می کنید، می توانید به راحتی از آنها در سایر زبان های برنامه نویسی نیز استفاده کنید.

4. یادگیری آن ساده است و در قالب قابل حمل ارائه می شود.

## **# استفاده از json.dump()**

برای استفاده از تابع json.dump () ابتدا ماژول json را وارد کنید. برای وارد کردن ماژول json ، از import json استفاده کنید. () json.dump به نوشتن داده ها در یک فایل JSON کمک می کند.

Syntax:

json.dump(data, file)

تابع () json.dump دو آرگومان دارد:

1. data: داده هایی که باید در یک فایل JSON نوشته شوند.

2. file: یک شیء فایل که می تواند برای ذخیره داده ها استفاده شود.

بیایید یک برنامه سریع برای ذخیره مجموعه ای از اعداد در یک فایل JSON ایجاد کنیم. برای ذخیره مجموعه اعداد ، از تابع json.dump () استفاده می کنیم:

import json

numbers = [10, 20, 30, 70, 191, 23] #create a set of numbers

filename = 'numbers.json' #use the file extension .json

with open(filename, 'w') as file\_object: #open the file in write mode

json.dump(numbers, file\_object) # json.dump() function to stores the set of numbers in

numbers.json file

در این برنامه، مجموعه اعداد را در numbers.json ذخیره می کنیم. پسوند .json نشان می دهد که فایل حاوی داده هایی با فرمت JSON است.

سپس به فایل در حالت 'w' (حالت نوشتن) دسترسی پیدا می کنیم تا بتوانیم داده ها را در یک فایل JSON نوشت. سرانجام ، تابع json.dump () مجموعه اعداد را در فایل number.json ذخیره می کند.

این برنامه خروجی ترمینال ندارد، اما وقتی پرونده number.json را باز می کنیم، داده های زیر را می بینیم:

[10, 20, 30, 70, 191, 23]

# استفاده از json.dumps()

متد () json.dumps می تواند برای تبدیل یک شی پایتون به یک رشته JSON استفاده شود.

Syntax:

json.dumps(data)

تابع () json.dumps یک پارامتر می گیرد ، یعنی داده هایی که باید به رشته JSON تبدیل شوند.

بیایید به مثال زیر نگاهی بیندازیم:

import json

data = {

'Name' : 'Felix',

'Occupation' : 'Doctor'

}

dict\_1 = json.dumps(data) # converting dictionary to JSON

print(dict\_1) # {'Name' : 'Felix','Occupation' : 'Doctor'}

## **# تفاوت بین dump و dumps**

1. متد dump () دو پارامتر (داده و فایل) ، در حالی که متد dumps () تنها یک پارامتر (داده) می گیرد.

2. برخلاف متد dumps، متد dump با عملیات فایل ترکیب می شود.

# استفاده از json.load()

ما از تابع json.load برای خواندن یک فایل JSON استفاده می کنیم. تابع json.load () یک آرگومان می گیرد که شیء فایل است.

Syntax:

json.load(file\_object)

فرض کنید، ما یک فایل JSON به نام student.json داریم که حاوی اشیاء JSON است.

{

"name": "Felix",

"Subjects": ["English", "Political Science"]

}

بیایید یک کد برای خواندن داده های ذخیره شده در پرونده student.json با استفاده از تابع json.load بنویسیم.

import json

with open(r,'student.json') as file\_object:

data = json.load(file\_object)

print(data) # {"name": "Felix", "Subjects": ["English", "Political Science"]}

تابع json.load () فایل JSON را تجزیه می کند و یک دیکشنری به نام data را برمی گرداند.

# استفاده از json.loads()

ما از متد json.loads () برای تجزیه یک رشته JSON و بازگشت یک شی پایتون مانند دیکشنری استفاده می کنیم. متد json.loads () محتویات فایل را به عنوان یک رشته می گیرد.

Syntax:

json.loads(json\_string)

Example:

import json

# JSON string:

dict\_1 = {

"Name": "Felix Maina",

"Contact Number": 0712345678,

"Email": "fely@gmail.com",

}

# parse dict\_1:

y = json.loads(dict\_1)

# the result is a Python dictionary:

print(y) #{ "Name": "Felix Maina", "Contact Number": 0712345678,"Email": "fely@gmail.com", }

در اینجا ، رشته dict\_1 با استفاده از متد () json.loads تجزیه می شود که یک دیکشنری به نام y را برمی گرداند.

توجه: تفاوت اصلی بین json.loads () و json.load () این است که json.loads () رشته ها را می خواند در حالی که json.load () برای خواندن فایل ها استفاده می شود.

## **# سریال سازی داده های JSON در پایتون**

Serialization فرآیند تبدیل یک نوع داده بومی به فرمت JSON است.

ماژول JSON یک شیء دیکشنری پایتون را به یک شی JSON تبدیل می کند. متدهای json.dump () و json.dumps () برای سریال سازی داده های پایتون به فرمت JSON استفاده می شود.

بیایید به یک مثال با استفاده از متد json.dump () نگاهی بیندازیم:

import json

# Data to be written

details = {

"name": "Felix Maina",

"years": 21,

"school": "Makerere"

}

# Serializing JSON and writing JSON file

with open("details.json", "w") as file\_object:

json.dump(details, file\_object) # {"name": "Felix Maina", "years": 21, "school": "Makerere"}

در اینجا، ما یک دیکشنری پایتون را به یک فایل فرمت JSON با نام details.json تبدیل می کنیم.

متد json.dumps () یک شی پایتون را به یک رشته JSON تبدیل می کند ، همانطور که در زیر نشان داده شده است:

import json

# Data to be written

details = {

"name": "Felix Maina",

"years": 21,

"school": "Makerere"

}

# Serializing JSON

json\_string = json.dumps( details )

print( json\_string ) #{"name": "Felix Maina", "years": 21, "school": "Makerere"}

## **# تغییر JSON به Object در پایتون**

Deserialization فرآیند تبدیل داده های JSON به یک نوع داده بومی است. در اینجا، ما داده های JSON را به یک دیکشنری در پایتون تبدیل می کنیم.

ما از متد json.loads () برای تغییر داده های JSON به یک شی پایتون استفاده می کنیم. روش json.load () نیز برای تغییر یک قالب فرمت JSON به یک شی پایتون استفاده می شود.

Example:

# importing the module

import json

# creating the JSON data as a string

data = '{"Name" : "Felix", "status" : "married"}'

print("data before deserailizing")

print(data) #json string

# deserailizing the data

h = json.loads(data)

print("data after deserailizing")

print(h) #python dictionary

Output:

# data before deserailizing

{"Name" : "Felix", "status" : "married"}

#data after deserailizing

{'status': 'married', 'Name': 'Felix'}

اجازه دهید یک فایل ایجاد کرده و نام آن را cars.json بگذارید. این فایل باید دارای اطلاعات زیر باشد:

{

"name": "Suzuki",

"year": 2001,

"model": "GDF10"

}

حالا بیایید با استفاده از تابع load () این فایل را از حالت طبیعی خارج کنیم:

import json

# opening the JSON file

data = open('cars.json','r')

print("Datatype before deserialization : ")

print(data) # prints the contents of the file

# deserailizing the data

h = json.load(data)

print("Datatype after deserialization : ")

print(h) # prints a python dictionary

=================================================

**underscore :**

اولین استفاده از underscoreها در پایتون زمانی هستش که شما میخواین از یک کلمه کلیدی به عنوان اسم یک آرگومان استفاده کنید. به کد زیر دقت کنید:

def show(name, class):

pass

همونطور که میدونید کلمه  class یک کلمه کلیدی رزرو شده پایتون هستش و شما نمیتونید ازش به عنوان اسم آرگومان استفاده کنید. به همین دلیل میتونید با اضافه کردن یک underscore به آخر کلمه اون رو از حالت رزرو خارج کنید.

def show(name, class\_):

pass

مصرف بعدی underscoreها در پایتون برای زمانی هستش که شما میخواید یه عضو از کلاس رو به شکل private یا protected در بیارید:

class Person:

name = 'amir' #public

\_age = 10 #protected

\_\_height = 170 #private

زمانی که قبل از اسم عضوی از کلاس یک underscore قرار بدین اون عضو به شکل protected خواهد بود. اگر دوتا underscore قرار بدین به شکل private خواهد بود.

مصرف بعدی زمانی هستش که شما به قبل و بعد از اسم یک متد در کلاس دوتا underscore اضافه کنید:

class Person:

def \_\_init\_\_(self):

pass

متدهایی که به این شکل اسم گذاری شوند به عنوان special method در پایتون شناخته میشوند. تمامی متدهای builtin پایتون به این شکل نامگذاری میشوند. شما به هیچ وجه نباید اسم متدهاتون رو به این شکل قرار بدین.

آخرین مصرف underscoreها در پایتون زمانی هست که شما مقدار متغیری که دارید براتون مهم نیست:

for i in range(10):

print('Hello')

مثال بالا رو اگه اجرا کنید مقدار Hello ده بار براتون اجرا میشه. اما اگه دقت کنید متغیر i که در حلقه for قرار دادید اصلا کاربرد نداره. پس به جاش میتونید به شکل زیر کار کنید:

for \_ in range(10):

print('Hello')

=================================================

**iterate :**

در این بخش میخوایم در مورد سه فعل مهم در پایتون صحبت کنیم: Iterable, Iterator, Iteration

هر سه این کلمات از کلمه Iterate مشتق شدن. در زبان انگلیسی کلمه Iterate به معنی تکرار کردن است.

اول مفهوم Iteration میگم. در پایتون به عمل تکرار کردن میگیم Iteration. ما در زبان پایتون دو نوع Iteration داریم. اولی حلقه for و دومی حلقه while. به مثال زیر دقت کنید:

for i in range(1, 11):

print(i)

در مثال بالا ما یک نمونه از Iteration رو داریم. پس به خود عمل تکرار کردن میگیم Iteration.

حالا Iterable رو میگیم. به آبجکت هایی که بتونیم روی اونها عمل تکرار رو انجام بدیم میگیم Iterable.

یا به عبارت دیگه به هر آبجکتی که بتونیم روش عمل Iteration رو انجام بدیم میگیم Iterable.

nums = [1, 2, 3, 4, 5]

for num in nums:

print(num)

در مثال بالا میبینید که ما داخل لیست nums حلقه زدیم و تونستیم به مقدارهاش دسترسی داشته باشیم. الآن لیست nums یک آبجکت Iterable هستش.

مورد بعدی Iterator هستش. به آبجکت هایی که بتونن آخرین وضعیت خودشون رو حفظ کنن میگیم iterator

آبجکت هایی که بشه روشون متد next رو اجرا کرد میتونن آخرین وضعیت خودشون رو حفظ کنن.

در حال حاضر متد next روی لیستی که در بالا ساختیم قابل اجرا نیستش.

print( dir( nums ) )

اگه کد بالا رو اجرا کنید تمام اتریبیوت ها و متدهایی که در nums در دسترس هستش رو به ما نشون میده:

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

در نتیجه ای که پایتون به ما نشون داده میبینید که متد \_\_next\_\_ وجود نداره.

اما به جاش متد \_\_iter\_\_ رو داریم که میتونه لیست nums رو به یک Iterator تبدیل کنه

inums = iter(nums)  
or  
inums = nums.\_\_iter\_\_()

در کد بالا با استفاده از متد iter لیست nums رو به یک Iterator تبدیل کردیم.

حالا اگه یک بار دیگه کد زیر رو اجرا کنید میبینید که متد next در دسترس هستش

print( dir( inums ) )

['\_\_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\_\_']

حالا میتونید از متد next استفاده کنید و به آیتم های داخل لیست inums به صورت تکی دسترسی پیدا کنید:

print( next(inums) )

print( next(inums) )

print( next(inums) )

print( next(inums) )  
or  
print( inums.\_\_next\_\_() )

حالا لیست inums یک Iterator هستش.