SEPTEMBER 29 – OCTOBER 5, 2019

المبادرة السعودية للمطورين

مسار الـPython

مشرفي المسار:

عبدالله عوده – انتصار النصار – رؤى كردي – لينا المصعبي

python™

ملاحظات قبل بدء الدروس:

- على المتدربين نشر كل يوم الجزئية التي تم كتابتها من النص البرمجي في الـ Github تحت Topic بعنوان saudidevorg

كما تم توضيحه في دروس الـ Github سابقاً

- على المتدربين نشر كل يوم مقدار التقدم وصورة لما تم تعلمه وتطبيقه على Twitter تحت الهاشتاقات:

#المبادرة_السعودية_للمطورين

#100_يوم_برمجة

#100DaysOfCode

تمنياتنا لك بالتوفيق

المبادرة السعودية للمطورين

اليوم الواحد والأربعون

الأصناف والكائنات في لغة البايثون

Python Classes and Objects

➢ Python Classes/Objects

الأصناف والكائنات في بايثون

لغة بايثون لغة برمجة تفسيرية، وهي لغة برمجة كائنية التوجّه

كُلّ شيء في لغة بايثون عبارة عن كائن، وليسَ كمتغير كما في اللغات البرمجية الأخرى.

Python is an object-oriented programming language. Almost everything in **Python** is an object, with its properties and methods.

A **Class** is like an object constructor, or a "**blueprint**" for creating objects.

الصنف عبارة عن كائن مشيّد، و في مفهوم برمجة الكائنات عند إنشاء صنف معين يسمى "النسخة الخام/النسخة الأصلية"
ثم ننشأ نسخة أو أكثر لهذا الصنف وهي ما تسمى بالكائنات، بالتالي نكون قد قمنا بتغييرات وتعديلات على النسخ

وليس الصنف الأساسي/الأصلي

حيث يكون الكود (متغيرات،مصفوفات، دوال، قوائم ..) مجموع بداخل صنف معين

ثم عند إنشاء كائن من هذا الصنف فإنه يمكننا تنفيذ مختلف العلميات داخل هذا الصنف

إن كنت مبرمج مبتدئ ولكن قررت تعلم بايثون

عليك الإطلاع والتعرّف على مفهوم البرمجة الكائنية التوجه OOP قبل إكمالك هذا الدرس

وهنا بعض الروابط التي قد تساعدك:

- برمجة كائنية التوجة : OOP - Part 1
- برمجة كائنية التوجة : OOP - Part 2
- What is OOP || ماهي البرمجة كائنية التوجه

➢ Create a Class

إنشاء صنف

To create a **class**, use the keyword **class**.

لإنشاء صنف جديد

نستخدم الكلمة المحجوزة class لتعريف الصنف، ثم نقوم بتسميته ثم وضع النقطتين الرأسيتين :

Example

Create a **class** named MyClass, with a property named x.

في المثال هنا

قمنا بإنشاء صنف بسيط باسم MyClass لا يقوم بفعل شيء

فقط قمنا بتعريف متغير عادي بداخله

```
class MyClass:
  x = 5

print(MyClass)
```

يمكنك أن تعرّف متغيرات في الصنف بشكل عادي

```
Python 3.7.2 (tags/v3.7.2:
(Intel)] on win32
Type "help", "copyright",
>>>
== RESTART:
<class '__main__.MyClass'>
>>>
```

نتيجة تشغيل الكود

لا يمكنك إنشاء كائن بدون صنف، لأن الكائن عبارة عن نسخة من صنف معين

➢ Create Object

إنشاء كائن من صنف

Now we can use the **class** named MyClass to create **objects**.

بعد ما قمت بإنشاء صنف جديد، يمكنك الآن إنشاء كائن من هذا الصنف

والكائن عبارة عن اسم كالمتغير تمامًا، وهو يمثّل نسخة مطابقة للصنف

Example

Create an **object** named p1, and print the value of x.

هنا قمنا بتعريف كائن وهو p1 من الصنف MyClass

```
class MyClass:
  x = 5

p1 = MyClass()
print(p1.x)
```

لإنشاء كائن نقوم بتعريف متغير ونقوم بإسناد اسم الصنف إليه ثم وضع القوسين ()

بعد إنشاء الكائن يمكنك الآن الوصول للمتغير x الذي بداخل الصنف MyClass

نكتب اسم الكائن ثم نقطة ثم اسم المتغير أو الشيء الذي تريد الوصول إليه (سواءا دالة أو متغير)

```
Python
(Intel
Type "
>>>
== RES
5
>>>
class MyClass:
  x = 5

p1 = MyClass()
print(p1.x)
```

تعريف الصنف

إنشاء متغير بداخل الصنف

إنشاء كائن لهذا الصنف

الوصول إلى المتغير وطباعة قيمته

نتيجة تشغيل الكود

بشكل عام الصنف يتكون من جزأين رئيسين :

١/خصائص Properties

٢/ وظائف Methods

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Properties = Attributes = Variables

Methods = Functions = Behaviour

➢ The __init__() Function

الدالة __init__()

The examples above are **classes** and **objects** in their simplest form and are not really useful in real life applications.

To understand the meaning of **classes** we have to understand the built-in __init__() function.

All **classes** have a function called __init__(), which is always executed when the **class** is being initiated. Use the __init__() function to assign values to **object** properties, or other operations that are necessary to do when the **object** is being created.

كل صنف في بايثون أوعند إنشائك واحدا، يملك هذا الصنف دوال /توابع جاهزة أهمها هي دالة __init__()

فهي تمكنك من إسناد أو تمرير قيم للخصائص مباشرة من خلال أقواس الدالة.

عندما تقوم بتعريف الدالة __init__() داخل الصنف، فكلما قمت بإنشاء كائن جديد لهذا الصنف (نسخة جديدة للصنف) فإن الصنف يقوم باستدعاء هذه الدالة تلقائيا

هذه الدالة تقوم بتنفيذ الكود الذي بداخلها عند إنشاء كائن من الصنف وتسمى أيضا بأنها تابع البناء constructor

Example

Create a **class** named Person, use the __init__() function to assign values for name and age.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

في هذا المثال

قمنا بتعريف صنف اسمه Person يحتوي على الدالة __init__()

عليك بتعريف الدالة كما تعلمت سابقا في دروس الدوال

وفي أقوس الدالة تم تمرير أسماء الخصائص التي نريد وضعها في الصنف

وإعطائها قيم أولية مباشرة عند إنشاء الكائنات من الصنف Person

هنا الدالة تحتوي على مُعاملين parameters وهما name و age

سيكون هذين المُعاملين عبارة عن خصائص للكائن بمجرد قيامنا بإنشاء نسخة جديدة للصنف Person

أنشأنا كائن وهو p1 ثم استدعينا دالة الطباعة لعرض القيم

```
Python
(Intel
Type "
>>>
== RES
John
36
>>>
```

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

نتيجة تشغيل الكود

Note: The __init__() function is called automatically every time the **class** is being used to create a new **object**.

دالة __init__() تعتبر تابعا خاصا من توابع الأصناف، وهو يعمل تلقائيا في كل مرة يتم فيها إنشاء نسخة جديدة من الصنف

➢ The self Parameter

الكلمة self كمُعامل

The **self** parameter is a reference to the current instance of the **class**, and is used to access variables that belongs to the **class**. It does not have to be named **self** , you can call it whatever you like, but it has to be the first parameter of any function in the **class**.

عند تعريف دالة بداخل الصنف، في بايثون يجب عليك وضع الكلمة self أو أي كلمة وذلك للإشارة أن الدالة تابعة للصنف

(المتعارف عليه بين مبرمجي بايثون هو استعمال self)

بحيث تكون كأول مُعامل parameter في الدالة، ثم تضع بعدها المُعاملات التي تريد

حتى لو أنك لا تريد أن تضع مُعاملات في الدالة، يلزمك وضع الكلمة self كمُعامل

وهذه الكلمة تعتبر كمؤشر للصنف نفسه، بحيث عن طريقها يمكنك الوصول إلى أي شيء تم تعريفة داخل الصنف

Example

Use the words **mysillyobject** and **abc** instead of **self**.

المُعامل هنا إجباري

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

المُعامل باسم abc إجباري

إنشاء كائن لهذا الصنف وتمرير قيمتين

استدعاء الدالة

قمنا بتعريف صنف باسم Person

وتعريف دالة __init__() تأخذ قيمتين

وتعريف دالة باسم myfunc تأخذ قيمة واحدة

```
Python 3.7.2 (tags/v3.7.
(Intel)] on win32
Type "help", "copyright"
>>>
== RESTART:
Hello my name is John
>>>
```

نتيجة تشغيل الكود

لاحظ أننا غيّرنا كلمة self إلى أسماء أخرى واستمر البرنامج بالعمل دون مشاكل

في لغات أخرى تُستعمل this بدلا عن self

أتممت درسك بنجاح!

روابط قد تهمك

Useful links

- Python class: Objects and classes
- Python Programming #14 - Object-Oriented Programming
- 4- Python is Objects
- #17 3 تعلم بايثون - Classes and Objects
- 30- Python OOP|| Basic Class
- 31- Python OOP|| Class Constructor
- Learn Python in Arabic #92 - انشاء كلاس مع الخصائص و الوظائف Create class attributes metho
- Learn Python in Arabic #93 - كائن من الكلاس Create instance object from class Python
- Learn Python in Arabic #94 - دالة بناء الكلاس constructor init in class Python
- Learn Python in Arabic #95 - دالة بناء بوسائط parameters constructor init in class Pyt
- #49 Python Tutorial for Beginners | Class and Object
- #50 Python Tutorial for Beginners | __init__ method
- #51 Python Tutorial for Beginners | Constructor, Self and Comparing Objects
- شرح الكلاس والاوبجكت - Classes and Objects 21 - تعلم البرمجة بلغة بايثون
- شرح - init 22 - تعلم البرمجة بلغة بايثون
- OOP : Create Class and Object
- OOP : Instance variables
- OOP : Parameter self
- OOP : self access to variables
- OOP : variables and functions

طبّق ما تعلمته في هذا الدرس

ولا تنسى مشاركتنا أكوادك

اليوم الثاني والأربعون

الأصناف والكائنات في لغة البايثون

Python Classes and Objects 2

➢ Object Methods توابع الكائن

Objects can also contain methods. Methods in **objects** are functions that belong to the **object**.

الكائنات أيضا لها توابع، التوابع هي عبارة دوال للكائن

Let us create a method in the **Person** **class**.

Example

Insert a function that prints a greeting and execute it on the **p1** **object**.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Python 3.7.2 (tags/v3.
(Intel)] on win32
Type "help", "copyrigh
>>>
== RESTART: C:\
Hello my name is John
>>>
```

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Note: The **self** parameter is a reference to the current instance of the **class**, and is used to access variables that belong to the class.

تذكر: تعتبر مؤشر للصنف نفسه، ويمكنك من خلالها الوصول إلى المتغيرات/الخصائص التي تم تعريفها داخل هذا الصنف

➢ Modify Object Properties

تعديل خصائص الكائن

You can modify properties on **objects** like this.

يمكنك تعديل/تغيير القيم

Example

Set the age of p1 to 40.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

p1.age = 40

print(p1.age)
```

هنا في هذا المثال

قمنا بتغيير قيمة age

```
Python class Person:
(Intel   def __init__(self, name, age):
Type "     self.name = name
>>>        self.age = age
== RES
40       def myfunc(self):
>>>        print("Hello my name is " + self.name)

         p1 = Person("John", 36)

         p1.age = 40

         print(p1.age)
```

نتيجة تشغيل الكود

➢ Delete Object Properties

حذف خصائص الكائن

يمكنك حذف خصائص/متغيرات الكائن عن طريق استخدام **del**

You can delete properties on **objects** by using the **del** keyword.

Example

Delete the **age** property from the **p1** **object**.

هنا في هذا المثال

قمنا بحذف الخاصيّة **age** من الكائن **p1**

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1.age

print(p1.age)
```

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for
>>>
== RESTART: (
Traceback (most recent call last):
  File "C:
, in <modul
    print(p1.age)
AttributeError: 'Person' object has no attribute 'age'
>>>
```

نتيجة تشغيل الكود

سيظهر لك **error** ولذلك لأن هذه الخاصية التي تبحث عنها لم تعد موجودة

➢ Delete Objects

حذف الكائن

You can delete **objects** by using the **del** keyword.

يمكنك حذف الكائن نفسه وذلك عن طريق استخدام **del**

Example

Delete the **p1** **object**.

هنا في هذا المثال

قمنا فقط بحذف الكائن p1

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1

print(p1)
```

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23
(Intel)] on win32
Type "help", "copyright", "credits" or "licer
>>>
== RESTART:
Traceback (most recent call last):
  File
, in <module>
    print(p1)
NameError: name 'p1' is not defined
>>>
```

نتيجة تشغيل الكود


```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1

print(p1)
```

أتممتَ درسك بنجاح!

تابع التقدّم

روابط قد تساعدك

Check the links below

- Delete Object or Properties in Python - Python Step By Step - 15

طبّق ما تعلمته في هذا الدرس

ولا تنسى مشاركتنا أكوادك

اليوم الثالث والأربعون

الوراثة في لغة البايثون

Python Inheritance

➢ Python Inheritance

الوراثة في بايثون

الوراثة تسمح لنا بتعريف صنف يرث جميع الدوال والخصائص الموجودة في صنف آخر

بمعنى آخر أي تضمين محتوى صنف في صنف آخر

Inheritance allows us to define a **class** that inherits all the methods and properties from another **class**.

الصنف الذي يورّث محتوياته لصنف آخر يسمى صنف الأب

Parent class is the **class** being inherited from, also called **base class**.

الصنف الذي يرث من صنف آخر يسمى صنف الابن

Child class is the **class** that inherits from another **class**, also called **derived class**.

➢ Create a Parent Class

إنشاء صنف الأب

Any **class** can be a **parent class**, so the syntax is the same as creating any other **class**

أي صنف يمكنه أن يكون صنف الأب

وطريقة كتابته وتعريفه كما كنا نعرّف الصنف في الدروس السابقة

Example

Create a **class** named **Person**, with **firstname** and **lastname** properties, and a **printname** method.

في هذا المثال

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

أنشأنا صنف باسم Peraon

يحتوي على الخاصيتين firstname و lastname

وعلى الدالة printname

```
Python 3.
(Intel)]
Type "hel
>>>
== RESTAR
John Doe
>>>
```

نتيجة تشغيل الكود

➢ Create a Child Class

إنشاء صنف الابن

To create a **class** that inherits the functionality from another **class**, send the **parent class** as a parameter when creating the **child class**.

لجعل صنف ما يرث المحتويات من صنف آخر:

نضع اسم صنف الأب ثم قوسين وبداخلهما اسم صنف الابن الذي نريده أن يرث من صنف الأب

Example

Create a **class** named **Student**, which will inherit the properties and methods from the **Person** **class**.

تم إنشاء صنف باسم Student الذي يمثل صنف الابن

والذي سيرث الصنف Person الذي يمثل صنف الأب

```
class Student(Person):
  pass
```

Now the **Student** **class** has the same properties and methods as the **Person** **class**.

الآن الصنف Student يحتوي على نفس الخصائص والدوال التي في الصنف Person

لاحظ: قم باستخدام الكلمة المفتاحية المحجوزة pass وذلك من أجل تعريف أصناف فارغة لا تحتوي على أي خصائص أو دوال

Note: Use the **pass** keyword when you do not want to add any other properties or methods to the **class**.

Example

Use the Student **class** to create an **object**, and then execute the printname method.

هنا في هذا المثال

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  pass

x = Student("Mike", "Olsen")
x.printname()
```

أنشأنا صنف باسم Person

يحتوي على الخاصيتين firstname و lastname

و يحتوي على الدالة printname

ثم تم إنشاء صنف باسم Student وجعلناه يرث من الصنف Person

ثم أنشأنا كائن x من الصنف Student مع تمرير القيمتين Mike و Olsen

ثم قمنا باستدعاء وتنفيذ الدالة printname

```
Python 3.7.2
(Intel)] on w
Type "help",
>>>
== RESTART: C
Mike Olsen
>>>
```

نتيجة تشغيل الكود

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  pass

x = Student("Mike", "Olsen")
x.printname()
```

➢ Add the __init__() Function

إضافة الدالة __init__() لصنف الابن

So far we have created a **child class** that inherits the properties and methods from its **parent**.

We want to add the **__init__()** function to the **child class** (instead of the **pass** keyword).

نريد استدعاء أو إضافة الدالة __init__() للصنف الابن بدلا ما كنا سابقا نضيف الكلمة المفتاحية pass

الصنف الابن سيرث كل شيء من الصنف الأب مباشرة، ماعدا الخصائص التي يتم تعريفها بداخل الدالة __init__()

وذلك لأن الدالة يتم استدعائها في الأصل لحظة إنشاء كائن من الصنف

Note: The **__init__()** function is called automatically every time the **class** is being used to create a new **object**.

تذكر أن الدالة __init__() يتم استدعائها تلقائيا عند إنشاء كائن من الصنف الذي يحتويها.

Example

Add the **__init__()** function to the **Student class**.

هنا قمنا بإضافة الدالة __init__() لصنف الابن Student

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

When you add the **__init__()** function, the **child class** will no longer inherit the parent's **__init__()** function.

عند إضافة الدالة __init__() لصنف الابن Student ، لن يرث صنف الابن دالة __init__() الخاصة بالأب

Note: The child's **__init__()** function **overrides** the inheritance of the parent's **__init__()** function.

overrides تعني تعريف الدالة التي ورثها الصنف الابن من الصنف الأب من جديد (إعادة الكتابة على الدوال)

To keep the **inheritance** of the parent's `__init__()` function, add a call to the parent's `__init__()` function:

إذا أردت المحافظة على وراثة الدالة `__init__()` من الأب، ويصبح الصنف الابن قادر على أن يعيد كتابة الدالة كما هي في صنف الأب ويتمكن من الوصول إليها، قم باستدعاء دالة `__init__()` الأب في صنف الابن

Example

انظر المثال التالي لتتضح لك الصورة

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

```
Python 3.7.2
(Intel)] on w
Type "help",
>>>
== RESTART: C
Mike Olsen
>>>
```

نتيجة تشغيل الكود

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

للوصول إلى الدالة المعرفة في صنف الأب

Now we have successfully added the `__init__()` function, and kept the **inheritance** of the **parent class**, and we are ready to add functionality in the `__init__()` function.

الآن تمت الوراثة والوصول إلى دالة `__init__()` الأب لصنف الأب.

بالتالي تستطيع الاستفادة من الدالة الموجودة في صنف الأب

وأيضا تستطيع إضافة وظائف أخرى جديدة في صنف الابن وليست موجودة في صنف الأب

مبرمج الغد!

أتممت درسك

روابط قد تهمك

Useful links

- Python tutorial Inheritance الوراثة
- Python tutorial - constructor with inheritance - الوراثة
- Python tutorial Override مفهوم الـ
- Python Programming #15 - Inheritance
- 33- Python OOP|| Class Inheritance الوراثة
- Learn Python in Arabic #106 - الوراثة inheritance in Python
- Learn Python in Arabic #107 - الوراثة عملي inheritance in Python

طبّق ما تعلمته في هذا الدرس

ولا تنسى مشاركتنا أكوادك

اليوم الرابع والأربعون

الوراثة في لغة البايثون

Python Inheritance 2

➢ Use the super() Function

استخدام الدالة super()

Python also have a **super()** function that will make the **child class** inherit all the methods and properties from its **parent**.

هذه الدالة تجعل صنف الابن يرث جميع الخصائص والدوال من صنف الأب مباشرة

Example

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

```
Python 3.7.
(Intel)] on
Type "help"
>>>
== RESTART:
Mike Olsen
>>>
```

نتيجة تشغيل الكود

By using the **super()** function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its **parent**.

دالة **super()** تمكنك من الوصول واستدعاء دالة **__init__()** الموجودة في صنف الأب

بشكل مباشر وتلقائي من الدالة **__init__()** الموجودة في صنف الابن

➢ Add Properties

إضافة خصائص جديدة لصنف الابن

Example

Add a property called **graduationyear** to the **Student** **class**.

في هذا المثال

قمنا بإضافة خاصيّة جديدة لصنف الابن Student وطباعة قيمتها

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019

x = Student("Mike", "Olsen")
print(x.graduationyear)
```

```
Python
(Intel)
Type "h
>>>
== REST
2019
>>>
```

نتيجة تشغيل الكود

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019

x = Student("Mike", "Olsen")
print(x.graduationyear)
```

In the example below, the year 2019 should be a variable, and passed into the Student **class** when creating student **objects**. To do so, add another parameter in the **__init__()** function.

Example

Add a year parameter, and pass the correct year when creating **objects**.

في هذا المثال

قمنا بإضافة مُعامل parameter جديد وهو year في دالة __init__() لصنف الابن Student

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Mike", "Olsen", 2019)
print(x.graduationyear)
```

نتيجة تشغيل الكود

```
Python 3
(Intel)]
Type "he
>>>
== RESTA
2019
>>>
```

➢ Add Methods

إضافة دوال جديدة لصنف الابن

Example

Add a method called welcome to the Student **class**.

في هذا المثال

قمنا بإضافة دالة جديدة لصنف الابن Student

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)

x = Student("Mike", "Olsen", 2019)
x.welcome()
```

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, D
(Intel)] on win32
Type "help", "copyright", "credits" or
>>>
== RESTART·
Welcome Mike Olsen to the class of 2019
>>>
```

نتيجة تشغيل الكود

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.grad

x = Student("Mike", "Olsen", 2019)
x.welcome()
```

If you add a method in the **child class** with the same name as a function in the **parent class**, the inheritance of the **parent** method will be **overridden**.

إذا قمت بإضافة دالة في صنف الابن تحمل نفس الاسم لدالة في صنف الأب، ستجد أنه سيتم تفعيل الـ **overridden** للدالة

أتممت درسك بنجاح!

واصل التعلّم

روابط قد تهمك

Useful links

- وراثة الأصناف في بايثون
- Python tutorial : Inheritance parent class
- #55 Python Tutorial for Beginners | Inheritance
- 34 - Python OOP|| method Overriding

طبّق ما تعلمته في هذا الدرس

ولا تنسى مشاركتنا أكوادك

اليوم الخامس والأربعون

المكرِّرات في لغة البايثون

Python Iterators

➢ Python Iterators

المكرِّرات في بايثون

An **iterator** is an object that contains a countable number of values.
An **iterator** is an object that can be iterated upon, meaning that you can traverse through all the values.

عبارة عن كائن يحتوي عدد محدد من القيم

يتم تخزين هذه القيم بالترتيب وراء بعضها البعض،

كلما وصلت إلى قيمة في هذا الكائن فإنك قادر للوصول إلى القيمة التالية التي تليها مباشرة

Technically, in **Python**, an **iterator** is an object which implements the **iterator protocol**, which consist of the methods **__iter__()** and **__next__()**

في بايثون الـ **iterator** عبارة عن كائن يطبّق برتوكول يسمى **iterator protocol**

يتم تطبيق هذا البروتوكول من خلال دالتين جاهزتين لذلك وهما **__iter__()** و **__next__()**

➢ Iterator vs Iterable

Lists, tuples, dictionaries, and sets are all **iterable** objects.
They are **iterable** containers which you can get an **iterator** from.
All these objects have a **iter()** method which is used to get an **iterator**.

القوائم، الصفوف، القواميس، والمجموعات كلها عبارة عن كائنات من نوع **iterable**

ولذلك لأنه يمكنك الوصول لأي عنصر فيهم مباشرة عن طريق الـ **index** أو **key** أي الوصول لقيم محددة

بعكس الـ **iterator** فإنه لا يمكن الوصول لقيم محددة فيه بشكل مباشر، يتم الوصول إلى قيمه بنفس الترتيب الذي تم تخزينهم فيه

نستخدم دالة جاهزة وهي **iter()** للكائنات من نوع **iterable** للحصول على كائن نوعه **iterator**

Example

في هذا المثال

Return an **iterator** from a **tuple**, and print each value.

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

قمنا بإنشاء صف mytuple ووضعنا فيه ثلاثة قيم

ثم أنشأنا iterator باستخدام الدالة iter() ووضعنا فيه قيم الكائن myit

ثم قمنا باستدعاء دالة next() ثلاث مرات وطباعة ما تعرضه في كل مرة

الدالة next() تعطيك القيمة التالية في مل مرة يتم استدعائها

هنا تقوم في كل مرة بترجيع القيمة التالية في الكائن myit

```
Python 3.
(Intel)]
Type "hel
>>>
== RESTAR
apple
banana
cherry
>>>
```

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

نتيجة تشغيل الكود

تم طباعة القيم واحدة تلو الأخرى

Even strings are **iterable** objects, and can return an **iterator**.

أيضا السلاسل النصية عبارة عن كائن **iterable**

Example

Strings are also **iterable** objects, containing a sequence of characters.

في هذا المثال

```
mystr = "banana"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

قمنا بتعريف متغير وأسندنا إليه قيمة نصية

ثم أنشأنا **iterator** باستخدام الدالة **iter()** ووضعنا فيه قيم الكائن myit

ثم قمنا باستدعاء دالة next() ست مرات لطباعة السلسلة الحرفية

في كل مرة تقوم الدالة next() بترجيع القيمة التالية في الكائن myit

```
Pytho
(Inte
Type
>>>
== RE
b
a
n
a
n
a
>>>
```

```
mystr = "banana"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

نتيجة تشغيل الكود

تم طباعة القيم واحدة تلو الأخرى

➢ Looping Through an Iterator

المرور على قيم المكرِّرات بواسطة الحلقة for

يمكنك استخدام الحلقة التكرارية for لعرض قيم الكائن iterable

We can also use a **for** loop to iterate through an **iterable** object.

Example

في هذا المثال

Iterate the values of a **tuple**.

```
mytuple = ("apple", "banana", "cherry")

for x in mytuple:
  print(x)
```

قمنا بإنشاء صف mytuple ووضعنا فيه ثلاث قيم

ثم قمنا باستخدام الحلقة for لطباعة القيم واحدة تلو الأخرى

```
Python 3
(Intel)
Type "he
>>>
== RESTA
apple
banana
cherry
>>>
```

```
mytuple = ("apple", "banana", "cherry")

for x in mytuple:
  print(x)
```

نتيجة تشغيل الكود

Example

Iterate the characters of a **string**.

المرور على جميع أحرف السلسلة النصية

في هذا المثال

```
mystr = "banana"

for x in mystr:
  print(x)
```

قمنا بتعريف متغير mystr يحتوي قيمة نصية

في كل دورة في الحلقة for يتم جلي حرف من النص وتخزينه في المتغير ومن ثم طباعته

```
Pytho mystr = "banana"
(Inte
Type for x in mystr:
>>>     print(x)
== RE
b
a
n
a
n
a
>>>
```

نتيجة تشغيل الكود

الحلقة for فعليا تنشأ كائن iterator وبالتالي فإن الحلقة تقوم بتنفيذ الدالة next() لترجيع القيمة التالية في كل دورة

The **for** loop actually creates an **iterator** object and executes the **next()** method for each loop.

➢ Create an Iterator

إنشاء المُكرِّر

لإنشاء كائن/صنف يعطيك iterator يجب أن تستخدم الدالتين __iter__() و __next__()

To create an **object/class** as an **iterator** you have to implement the methods **__iter__()** and **__next__()** to your **object**.

كما قد تعلمت في دروس الأصناف والكائنات سابقا، بأن لها دالة __init__() تعطي قيم أولية عند إنشاء الكائن للصنف

As you have learned in the **Python Classes/Objects** chapter, all **classes** have a function called **__init__()**, which allows you do some initializing when the object is being created.

الدالة __iter__() تعمل مثل الدالة __init__() في الأصناف، لكن أولًا يجب أن تقوم الدالة بترجيع كائن iterator نفسه

The **__iter__()** method acts similar, you can do operations (initializing etc.), but must always return the **iterator** object itself.

الدالة __next__() أيضا تعمل مثل دالة الـ __init__() في الأصناف، لكن يجب أن تقوم الدالة بترجيع القيمة التالية

The **__next__()** method also allows you to do operations, and must return the next item in the sequence.

Example

في المثال التالي

Create an **iterator** that returns numbers, starting with **1**, and each sequence will increase by one (returning **1**,**2**,**3**,**4**,**5** etc.)

قمنا بإنشاء صنف يعطينا iterator يقوم بترجيع أرقام، وذلك ابتداء من الرقم 1 و في كل مرة تكون الزيادة بواحد 1

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    x = self.a
    self.a += 1
    return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

القيمة الأولية للمتغير a

في كل مرة نقوم باستدعاء الدالة next() ستكون قيمة المتغير a مضافا عليها واحد

التحويل إلى iterator وذلك باستخدام الدالة iter()

هنا قمنا بعرض القيمة التالية التي سيقوم بإرجاعها الكائن وذلك عن طريق الدالة next()

```
Pytho class MyNumbers:
(Inte   def __iter__(self):
Type      self.a = 1
>>>       return self
== RE
1       def __next__(self):
2         x = self.a
3         self.a += 1
4         return x
5
>>> myclass = MyNumbers()
    myiter = iter(myclass)

    print(next(myiter))
    print(next(myiter))
    print(next(myiter))
    print(next(myiter))
    print(next(myiter))
```

نتيجة تشغيل الكود

➢ StopIteration

العبارة StopIteration

The example above would continue forever if you had enough **next()** statements, or if it was used in a **for** loop. To prevent the **iteration** to go on forever, we can use the **StopIteration** statement.

في المثال السابق الصنف سيعطينا iterator لا نهاية له لو قمنا باستدعاء الدالة next() بشكل كافٍ، أو لو قد استخدمنا الحلقة for ، لذلك نستخدم العبارة StopIteration لمنع iteration أن تكون بلا نهاية

In the **__next__()** method, we can add a terminating condition to raise an error if the **iteration** is done a specified number of times.

في دالة __next__() يمكن أن نضيف شرطا ليمنع هذا الخطأ من الحدوث، في حال وصل الـ iterator إلى نهايته

Example

Stop after 20 **iterations**.

في هذا المثال يتوقف الـ iterator عند العدد 20

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
  print(x)
```

```
Python
(Intel
Type "
>>>
== RES
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
>>>
```

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
  print(x)
```

نتيجة تشغيل الكود

رائع!

أتممت درسك الأخير لهذا الأسبوع

روابط قد تهمك

Useful links

- في بايثون Iterators المكرّرات
- #61 Python Tutorial for Beginners | Iterator
- 23- Python|| iterator For Loop العبارات التكرارية

طبّق ما تعلمته في هذا الدرس

ولا تنسى مشاركتنا أكوادك

اليوم السادس والأربعون

&

اليوم السابع والأربعون

تحدي الأسبوع (يتم حله ورفعه على Github)

١/ قُم بعمل صنف **Class** باسم **Library** ثم قُم بعمل كائن **Object** من الصنف السابق

وقُم بتعيين عنصرين داخل هذا الصنف وهما: **shelf** ، **book**

حيثُ أن الـ **book** متغير يحفظ قيمة عدد الكتب في المكتبة مثلًا: **300**

والـ **shelf** متغير يحفظ قيمة عدد رفوف الكتب في المكتبة لنقل مثلا: **45**

٢/ -بعد عمل فقرة ١- قم بإنشاء صنف ابن من الصنف الأب **Library** وتسميته **science_section**

بحيث يأخذ كل خصائص الصنف الأب ثم أضف عليه العنصر **name** وأعطه الاسم " **Physics books** "

ثم اطبع محتوى كائن الصنف الابن.

٣/ أخيرا .. -وإكمالًا للفقرة ٢- قُم بتغيير عدد الكتب والرفوف في الصنف الابن إلى: **20** كتاب و **4** رفوف، ثم اطبع النتيجة!

بالتوفيق

انتظرنا في دروس الأسبوع القادم