What is PEP 8 ?

**PEP** stands for Python Enhancement Proposal. PEP 8 is a coding convention, a set of recommendations, about how to write your Python code more readable.

In other words, PEP 8 is a document that gives coding conventions for the Python code comprising the standard library in the main Python distribution. One of Guido's (author of Python) key insights is that code is read much more often than it is written.

The guidelines **(PEP 8)** provided [**here**](https://www.python.org/dev/peps/pep-0008/#introduction) are intended to improve the readability of code and make it consistent across the wide spectrum of Python code.

A style guide is about consistency. Consistency with this style guide is **important**. Consistency within a project is **more important**. Consistency within one module or function is the **most important**.

However, know when to be inconsistent. Sometimes style guide recommendations aren't just applicable. When in doubt, use your best judgment. Look at other examples and decide what looks best. And don't hesitate to ask someone else.

The main idea of PEP 8 is to use the same code style for all Python projects as if they were written by the same programmer. PEP 8, even for beginners, assures that it will easily understand the code written by any other developer.

Q: What is PEP 8?  
A: PEP stands for Python Enhancement Proposal. PEP 8 is a coding convention, a set of recommendation, about how to write your Python code more readable. In other words, PEP 8 is a document that gives coding conventions for the Python code comprising the standard library in the main Python distribution.

PEP8 nedir?  
PEP, Python Geliştirme Önerisi anlamına gelir. PEP 8, Python kodunuzu nasıl daha okunaklı yazacağınızla ilgili bir kodlama kuralı, bir dizi öneridir.  
  
Başka bir deyişle, PEP 8, ana Python dağıtımında standart kitaplığı içeren Python kodu için kodlama kuralları veren bir belgedir. Guido'nun (Python yazarı) önemli görüşlerinden biri, kodun yazıldığından çok daha sık okunmasıdır.  
  
Burada sağlanan yönergeler (PEP 8) kodun okunabilirliğini iyileştirmeyi ve geniş Python kodu yelpazesinde tutarlı hale getirmeyi amaçlamaktadır.  
  
  
  
Bir stil rehberi tutarlılıkla ilgilidir. Bu stil kılavuzuyla tutarlılık önemlidir. Bir proje içindeki tutarlılık daha önemlidir. Bir modül veya fonksiyon içindeki tutarlılık en önemlisidir.  
  
Ancak, ne zaman tutarsız olmanız gerektiğini bilin. Bazen stil rehberi önerileri sadece uygulanabilir değildir. Şüphe duyduğunuzda, en iyi kararınızı kullanın. Diğer örneklere bakın ve neyin en iyi göründüğüne karar verin. Ve başka birine sormaktan çekinmeyin.  
  
PEP 8'in ana fikri, tüm Python projeleri için aynı programcı tarafından yazılmış gibi aynı kod stilini kullanmaktır. PEP 8, yeni başlayanlar için bile, başka herhangi bir geliştirici tarafından yazılan kodu kolayca anlayacağını garanti eder.  
  
S: PEP 8 nedir?  
C: PEP, Python Geliştirme Önerisi anlamına gelir. PEP 8, Python kodunuzu nasıl daha okunaklı yazacağınızla ilgili bir kodlama kuralı, bir dizi öneridir. Başka bir deyişle, PEP 8, ana Python dağıtımında standart kitaplığı içeren Python kodu için kodlama kuralları veren bir belgedir.

Some Important PEP 8 Rules

We will show you some important PEP 8 traditional rules that you can follow.

* Limit all lines to a maximum of **79 characters**. For flowing long blocks of text with fewer structural restrictions (docstrings or comments), the line length should be limited to **72 characters**. During this course, we will learn some ways of reducing the length of lines.
* **Spaces**are the preferred indentation method. **Tabs** should be used solely to remain consistent with code that is already indented with tabs. Python 3 disallows mixing the use of tabs and spaces for indentation.
* Avoid extraneous **whitespaces** in the following situations:

Immediately inside parentheses, brackets or braces :

**YES** : spam(meat[1], {milk: 2}) , **NO** : spam( meat[ 1 ], { milk: 2 } )

Between a trailing comma and a following close parenthesis :

**YES** : df[0,] or foo = (2,) , **NO** : df[0, ] or foo = (2, )

Immediately before a comma, semicolon, or colon :

**YES** : if y == 3: print x, y; x, y = y, x , **NO** : if y == 3 : print x , y ; x , y = y , x

Immediately before the open parenthesis that starts the argument list of a function call:

**YES** : print('peace') , **NO** : print ('peace')

More than one space around an assignment (or other) operator to align it with another:

| **YES** | **NO** |
| --- | --- |
| x = 3 | x =mmmmm3 |
| y = 4 | y =mmmmm4 |
| long\_vars = 5 | long\_vars = 5 |

* Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing: e.g. a backslash followed by a space and a newline does not count as a line continuation marker.
* Always surround these binary operators with a single space on either side: assignment (=), augmented assignment (+=, -=, etc.), comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not), Booleans (and, or, not).

Failure to follow the basic rules of PEP 8 does not make your program wrong or unable to work. In the near future, you will learn a lot about Python and become a more skilled programmer, but it will always be important to follow the code style.

There's nothing to worry about following PEP 8. You don't need to learn the traditional PEP 8 rules all at once right away. When you need it, you can open and read it now and then. We will also show you some PEP 8 conventions throughout this course.

Bazı Önemli PEP 8 Kuralları  
Size izleyebileceğiniz bazı önemli PEP 8 geleneksel kurallarını göstereceğiz.  
  
Tüm satırları maksimum 79 karakterle sınırlayın. Daha az yapısal kısıtlamaya sahip uzun metin blokları (belge dizileri veya yorumlar) için satır uzunluğu 72 karakterle sınırlandırılmalıdır. Bu kurs sırasında, çizgilerin uzunluğunu azaltmanın bazı yollarını öğreneceğiz.  
  
Boşluklar tercih edilen girinti yöntemidir. Sekmeler yalnızca, zaten sekmelerle girintili olan kodla tutarlı kalmak için kullanılmalıdır. Python 3, girinti için sekme ve boşluk kullanımının karıştırılmasına izin vermez.  
  
Aşağıdaki durumlarda gereksiz boşluklardan kaçının:  
  
Hemen parantez, parantez veya parantez içinde:  
  
EVET : spam(et[1], {süt: 2}) , HAYIR : spam( et[ 1 ], { süt: 2 } )  
  
Sondaki virgül ile aşağıdaki yakın parantez arasında:  
  
EVET : df[0,] veya foo = (2,) , HAYIR : df[0, ] veya foo = (2, )  
  
Virgül, noktalı virgül veya iki nokta üst üste işaretinden hemen önce:  
  
EVET : eğer y == 3: x, y yazdır; X, y = y, x , NO : y == 3 ise: x , y yazdır; X , y = y , x  
  
Bir işlev çağrısının bağımsız değişken listesini başlatan açık parantezden hemen önce:  
  
EVET : yazdır('barış') , HAYIR : yazdır ('barış')  
  
Bir atama (veya başka bir) operatörün çevresinde, onu bir başkasıyla hizalamak için birden fazla boşluk:  
  
EVET HAYIR  
X = 3x =mmmm3  
Y = 4y =mmmm4  
Long\_vars = 5long\_vars = 5

Herhangi bir yerde boşluk bırakmaktan kaçının. Genellikle görünmez olduğu için kafa karıştırıcı olabilir: ör. Ters eğik çizgi, ardından boşluk ve yeni satır, satır devam işareti olarak sayılmaz.  
  
Bu ikili operatörleri her zaman her iki tarafta tek bir boşlukla çevreleyin: atama (=), artırılmış atama (+=, -=, vb.), karşılaştırmalar (==, , , , !=, , =, =, in, not in , is, is not), Booleans (ve, or, not).  
  
PEP 8'in temel kurallarına uymamak, programınızı yanlış veya çalışamaz hale getirmez. Yakın gelecekte Python hakkında çok şey öğrenecek ve daha yetenekli bir programcı olacaksınız, ancak kod stilini takip etmek her zaman önemli olacak.  
  
PEP 8'i takip etmek konusunda endişelenecek bir şey yok. Geleneksel PEP 8 kurallarının hepsini bir anda öğrenmenize gerek yok. İhtiyacınız olduğunda, ara sıra açıp okuyabilirsiniz. Ayrıca bu kurs boyunca size bazı PEP 8 kurallarını da göstereceğiz.

Comments and Docstrings

Introduction

When writing a program, you will need to add explanatory notes to others or even yourself. The longer you write lines, the better you will understand the necessity of this. We can add these explanatory notes to our program as 'comment' or as 'docstring' in more detail.

Tanıtım  
Bir program yazarken, başkalarına ve hatta kendinize açıklayıcı notlar eklemeniz gerekecektir. Ne kadar uzun satırlar yazarsanız, bunun gerekliliğini o kadar iyi anlayacaksınız. Bu açıklayıcı notları daha detaylı olarak 'yorum' veya 'doküman dizisi' olarak programımıza ekleyebiliriz.

**Comments**

**Comments** are used to explain code when the basic code itself isn't clear. Python ignores comments, and so will not execute code in there, or raise syntax errors for plain English sentences.

There are three types of commenting methods. These are :

* **Single-line comments** begin with the hash character 👉🏻# and are terminated by the end of the line. 👉🏻# sign converts all subsequent characters to the comment form that Python does nothing.

input :

# This is a single line comment

* **Inline comments** also begin with hash character 👉🏻# and start from the end of a code line.

input :

print('the cosmos has no superiority to chaos') # This is an inline comment

output :

the cosmos has no superiority to chaos

* **Multi-line comments** basically consist of multiple comment lines.

input :

print(3 + 4)

# This is the multi-line comment, line-1

# This is the multi-line comment, line-2

# This is the multi-line comment, line-3

output :

7

**💡Tips:**

* To begin with, after 👉🏻 **#** there should be one space, and in the inline comments, there should be at least two spaces between the end of the code and 👉🏻 **#**.
* A comment is not a piece of code. It should be short. It's better to split a long comment into multiple lines. You have to add 👉🏻 **#** at the beginning of each new line.

Apart from the well-readable syntax itself, in writing Python programs, there are other important things that contribute to understandability of your program. We assume that you are familiar with comments and how they help in understanding codes.

In the real programming world, comments become especially important as the program gets bigger and more complicated. Without using them, things may get confusing for other developers who see your code for the first time. It may get confusing even for you within a couple of months after writing the program.

**⚠️Avoid ! :**

* More comments don't necessarily need to be better. If code is self-explanatory, comments are unnecessary.
* Do not make unnecessary comments. Usually, comments should answer the question **why** as opposed to **what**.
* When necessary, **update your comment**. Be sure that your comments will not be in contradiction to the code.

Q: What are the comments and how do you write it in Python?  
A: Comments are used to explain code when the basic code itself isn't clear. Python ignores comments, and so will not execute code in there, or raise syntax errors for plain English sentences. Comments in Python start with a # character. '#' character converts all subsequent characters to the comment form that Python does nothing.  
  
# this is a single line comment  
  
print("Hello World!") # this is an inline comment

- Interview Q&A

### Yorumlar Yorumlar, temel kodun kendisi net olmadığında kodu açıklamak için kullanılır. Python yorumları yok sayar ve bu nedenle orada kod yürütmez veya düz İngilizce cümleler için sözdizimi hataları oluşturmaz. Üç tür yorumlama yöntemi vardır. Bunlar : Tek satırlık yorumlar 👉🏻# hash karakteri ile başlar ve satırın sonunda sonlandırılır. 👉🏻# işareti, sonraki tüm karakterleri Python'un hiçbir şey yapmadığı yorum formuna dönüştürür. Giriş: # Bu tek satırlık bir yorumdur Satır içi yorumlar ayrıca 👉🏻# karma karakteri ile başlar ve bir kod satırının sonundan başlar. Giriş: Print('kozmosun kaosa üstünlüğü yoktur') # Bu bir satır içi yorumdur Çıktı : Kozmosun kaosa üstünlüğü yoktur Çok satırlı yorumlar temel olarak birden çok yorum satırından oluşur. Giriş: Yazdır(3 + 4) # Bu çok satırlı yorum, satır-1 # Bu çok satırlı yorum, satır-2 # Bu çok satırlı yorum, satır-3 Çıktı : 7 💡İpuçları: Başlangıç ​​olarak, 👉🏻#'dan sonra bir boşluk, satır içi yorumlarda ise kodun sonu ile 👉🏻 # arasında en az iki boşluk olmalıdır. Yorum bir kod parçası değildir. Kısa olmalı. Uzun bir yorumu birden çok satıra bölmek daha iyidir. Her yeni satırın başına 👉🏻# eklemelisiniz. Python programlarını yazarken, iyi okunabilir sözdiziminin yanı sıra, programınızın anlaşılabilirliğine katkıda bulunan başka önemli şeyler de vardır. Yorumlara ve kodların anlaşılmasına nasıl yardımcı olduklarına aşina olduğunuzu varsayıyoruz. Gerçek programlama dünyasında, program büyüdükçe ve karmaşıklaştıkça yorumlar özellikle önemli hale gelir. Bunları kullanmadan, kodunuzu ilk kez gören diğer geliştiriciler için işler kafa karıştırıcı olabilir. Programı yazdıktan sonraki birkaç ay içinde sizin için bile kafa karıştırıcı olabilir. ⚠️ Kaçının ! : Daha fazla yorum mutlaka daha iyi olmak zorunda değildir. Kod kendini açıklayıcıysa, yorumlar gereksizdir. Gereksiz yorumlar yapmayın. Genellikle yorumlar, neyin aksine neden sorusuna cevap vermelidir. Gerektiğinde yorumunuzu güncelleyin. Yorumlarınızın koda aykırı olmayacağından emin olun. S: Yorumlar nelerdir ve bunu Python'da nasıl yazıyorsunuz? C: Temel kodun kendisi net olmadığında, yorumlar kodu açıklamak için kullanılır. Python yorumları yok sayar ve bu nedenle orada kod yürütmez veya düz İngilizce cümleler için sözdizimi hataları oluşturmaz. Python'da yorumlar # karakteriyle başlar. '#' karakteri, sonraki tüm karakterleri Python'un hiçbir şey yapmadığı yorum formuna dönüştürür. # bu tek satırlık bir yorumdur Print('Merhaba Dünya!') # bu bir satır içi yorumdur Docstrings

We have to say at the beginning that you will not learn to create and write docstrings in this course. Only what we will show you is what docstrings are and how we will call and display it.

**Docstrings** are - unlike regular comments - stored as an attribute of the function or the module they document, meaning that you can access them programmatically. Docstring runs as an explanatory text of codes and it should be written between triple quotes. Like: """docstring""".

**💡Tips:**

* You don't need to learn or know; 'what the **function** and the **module** are?' for now.
* We will show you these topics in the Python Basics Plus Course.

Although it is not mandatory to learn, for now, you can consider the definitions of these terms below:

* We have briefly mentioned its meaning before. A **function** is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions.
* A **module** is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes, and variables. A module can also include runnable code.

Normally, when we want to call docstring of a function or module to read, we will use **\_\_doc\_\_** (the keyword doc enclosed by double underscores) syntax. See the example below :

Here is an example:

input :

def function(): # Don't be confused, we use 'def()' to create a function.

# You will see it in the next lessons.

"""

Hi, I am the docstring of this code.

If you need any information about this function or module, read me.

It can help you understand how the module or function works.

"""

print(function.\_\_doc\_\_)

output :

Hi, I am the docstring of this code.

If you need any information about this function or module, read me.

It can help you understand how the module or function works.

Let's see the docstring of print() function:

input :

print(print.\_\_doc\_\_)

output :

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

Q: What is docstring in Python?  
A: Docstrings are - unlike regular comments - stored as an attribute of the function or the module they document, meaning that you can access them programmatically. Docstring runs as an explanatory text of codes and it should be written between triple quotes.

- Interview Q&A

Belge Dizileri  
Bu kursta belge oluşturmayı ve yazmayı öğrenmeyeceğinizi baştan söylemeliyiz. Sadece size göstereceğimiz şey, dokümantasyonların ne olduğu ve onu nasıl çağırıp göstereceğimizdir.  
  
Belge dizileri - normal yorumların aksine - belgeledikleri işlevin veya modülün bir özelliği olarak saklanır, yani bunlara programlı olarak erişebilirsiniz. Docstring, açıklayıcı bir kod metni olarak çalışır ve üçlü tırnak işaretleri arasında yazılmalıdır. Gibi: '''belge dizisi'''.  
  
💡İpuçları:  
Öğrenmenize veya bilmenize gerek yok; 'fonksiyon ve modül nedir?' Şimdilik.  
Bu konuları Python Temelleri Artı Kursunda göstereceğiz.  
Öğrenmek zorunlu olmasa da şimdilik bu terimlerin tanımlarını aşağıda inceleyebilirsiniz:  
  
Daha önce kısaca anlamından bahsetmiştik. İşlev, ilgili tek bir eylemi gerçekleştirmek için kullanılan organize, yeniden kullanılabilir bir kod bloğudur. İşlevler, uygulamanız için daha iyi modülerlik ve yüksek derecede kod yeniden kullanımı sağlar. Bildiğiniz gibi Python size print(), vb. gibi birçok yerleşik işlev sunar, ancak kendi işlevlerinizi de oluşturabilirsiniz.  
  
Modül, bağlayabileceğiniz ve başvurabileceğiniz keyfi olarak adlandırılmış niteliklere sahip bir Python nesnesidir. Basitçe, bir modül Python kodundan oluşan bir dosyadır. Bir modül fonksiyonları, sınıfları ve değişkenleri tanımlayabilir. Bir modül çalıştırılabilir kod da içerebilir.  
  
Normalde, okumak için bir işlevin veya modülün docstring'ini çağırmak istediğimizde, \_\_doc\_\_ (çift alt çizgi içine alınmış doc anahtar sözcüğü) sözdizimini kullanırız. Aşağıdaki örneğe bakın:  
  
İşte bir örnek:  
  
Giriş:  
  
Def function(): # Kafanız karışmasın, bir fonksiyon yaratmak için 'def()' kullanıyoruz.  
# İlerleyen derslerde göreceksiniz.  
'''  
Merhaba, ben bu kodun dokümanıyım.  
Bu işlev veya modül hakkında herhangi bir bilgiye ihtiyacınız varsa, beni okuyun.  
Modül veya işlevin nasıl çalıştığını anlamanıza yardımcı olabilir.  
'''  
Yazdır(işlev.\_\_doc\_\_)  
Çıktı :  
  
Merhaba, ben bu kodun dokümanıyım.  
Bu işlev veya modül hakkında herhangi bir bilgiye ihtiyacınız varsa, beni okuyun.  
Modül veya işlevin nasıl çalıştığını anlamanıza yardımcı olabilir.  
Print() fonksiyonunun dokümanını görelim:  
  
Giriş:  
  
Yazdır(yazdır.\_\_doc\_\_)  
Çıktı :  
  
Print(değer, ..., sep=' ', bitiş='  
', dosya=sys.stdout, flush=Yanlış)  
  
Değerleri bir akışa veya varsayılan olarak sys.stdout'a yazdırır.  
İsteğe bağlı anahtar kelime bağımsız değişkenleri:  
Dosya: dosya benzeri bir nesne (akım); Geçerli sys.stdout için varsayılandır.  
Sep: değerler arasına eklenen dize, varsayılan olarak bir boşluk.  
End: son değerden sonra eklenen dize, varsayılan olarak yeni bir satır.  
Yıkama: akışın zorla boşaltılıp yıkanmayacağı.  
S: Python'da doküman dizisi nedir?  
C: Belge dizileri - normal yorumların aksine - belgeledikleri işlevin veya modülün bir özelliği olarak saklanır, yani bunlara programlı olarak erişebilirsiniz. Docstring, açıklayıcı bir kod metni olarak çalışır ve üçlü tırnak işaretleri arasında yazılmalıdır.

Naming Variables - Değişkenleri Adlandırma

General Description

As you know, each variable has a unique name that distinguishes it from others. Giving a good name to a variable may not be as simple as it sounds.

A Python **variable** is a reserved memory location to store values. In other words, a variable gives data to the computer for processing. We will discuss variables in detail in the next lessons.

**💡Tips:**

* Remember, a nice and meaningful naming of variables is a skill that can be gained over time. Of course, you also need to be familiar with PEP 8 traditional rules.

Expert programmers care much for naming the variables well to make their codes easy to understand. It is important because programmers spend a lot of time reading and understanding code written by other programmers.

The convention of naming is optional. You can use any names you like but it is useful to follow the convention so that someone (including you) knows what you have written.

Genel açıklama  
Bildiğiniz gibi, her değişkenin onu diğerlerinden ayıran benzersiz bir adı vardır. Bir değişkene iyi bir isim vermek göründüğü kadar basit olmayabilir.  
  
Python değişkeni, değerleri depolamak için ayrılmış bir bellek konumudur. Başka bir deyişle, bir değişken bilgisayara işlenmek üzere veri verir. Değişkenleri sonraki derslerde detaylı olarak tartışacağız.  
  
💡İpuçları:  
Unutmayın, değişkenlerin güzel ve anlamlı bir şekilde adlandırılması zamanla kazanılabilecek bir beceridir. Elbette, PEP 8 geleneksel kurallarına da aşina olmanız gerekir.  
Uzman programcılar, kodlarının anlaşılmasını kolaylaştırmak için değişkenleri iyi adlandırmaya çok önem verirler. Bu önemlidir çünkü programcılar diğer programcılar tarafından yazılan kodu okumak ve anlamak için çok zaman harcarlar.  
  
Adlandırma kuralı isteğe bağlıdır. İstediğiniz herhangi bir ismi kullanabilirsiniz, ancak birinin (siz dahil) yazdıklarınızı bilmesi için kuralı takip etmek yararlıdır.

Conventional (PEP 8) Naming Rules

If variables have poor names, even your own code may seem unclear to you in a couple of months. Now let's learn how to choose good names for our variables in accordance with PEP 8 rules:

* Choose lowercase words and use underscore to split the words:
* **price = [22, 44, 66]**,
* **low\_price = 12.00**
* Do not use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single-character variable names. In some fonts, these characters are indistinguishable from the numerals one and zero. If you want to use 'l', use 'L' instead.
* **l = 'It is not correct use'**,
* **O = "It's also incorrect use"**

**⚠️Avoid ! :**

* Do not use specific Python keywords (name of a function or phrase) as a name, like sum, max, min, in, or, for, etc.
* Use a sensible name. The variable name needs to be legible and meaningful and explain to the reader what types of values will be stored in it.
* **figures = 'this is better'**,
* **f = 'it is not meaningful'**
* Don't choose too common names. Use a name to describe the meaning of the variable. However, try to limit it to no more than 3 words.
* If the word you intend to choose is long, try to find the most common and expected short form to make it easy to predict later.

| **Variable to be named** | **Sample of Good name** | **Sample of Bad name** | **Why bad?** |
| --- | --- | --- | --- |
| Cleaned Data | cleaned\_data | cdat | it doesn't make sense enough. |
| Indexes of the Clear Application Syntaxes | clr\_app\_syntx | ix\_app\_syntax | it doesn't make sense enough. |
| Customer Information of the Bank Accounts | customer\_bank\_info | costomer\_info\_bank\_account | it's too wordy. |

Q: Which of the following is an invalid statement?  
A:  
a) x, y, z = 1, 22, 333  
b) x\_y\_z = 1\_234\_567  
c) xyz = 1234567  
**d) x y z = 111 222 333**  
  
Spaces are not allowed in variable names

Değişkenleri Adlandırma  
Yapılacaklar: Aktivitenin sonuna kadar gidin  
✔Genel Açıklama  
  
✔Geleneksel (PEP 8) Adlandırma Kuralları  
  
Geleneksel (PEP 8) Adlandırma Kuralları  
Değişkenlerin adları kötüyse, kendi kodunuz bile birkaç ay içinde size belirsiz görünebilir. Şimdi değişkenlerimiz için PEP 8 kurallarına göre nasıl iyi isimler seçeceğimizi öğrenelim:  
  
Küçük harfli kelimeler seçin ve kelimeleri bölmek için alt çizgi kullanın:  
Fiyat = [22, 44, 66],  
Düşük\_fiyat = 12.00  
Tek karakterli değişken adları olarak 'l' (küçük harf el), 'O' (büyük harf oh) veya 'I' (büyük harf göz) karakterlerini kullanmayın. Bazı yazı tiplerinde bu karakterler bir ve sıfır rakamlarından ayırt edilemez. 'l' kullanmak istiyorsanız, bunun yerine 'L' kullanın.  
L = 'Doğru kullanım değil',  
O = 'Ayrıca yanlış kullanım'  
⚠️ Kaçının ! :  
Toplam, maks, min, in veya, for, vb. gibi belirli Python anahtar sözcüklerini (bir işlevin veya tümcecik adı) ad olarak kullanmayın.  
Mantıklı bir ad kullanın. Değişken adı okunaklı ve anlamlı olmalı ve okuyucuya içinde ne tür değerlerin saklanacağını açıklamalıdır.  
Rakamlar = 'bu daha iyi',  
F = 'anlamlı değil'  
Çok yaygın isimler seçmeyin. Değişkenin anlamını açıklamak için bir ad kullanın. Ancak, en fazla 3 kelime ile sınırlamaya çalışın.  
  
Seçmeyi düşündüğünüz kelime uzunsa, daha sonra tahmin etmeyi kolaylaştırmak için en yaygın ve beklenen kısa formu bulmaya çalışın.  
  
Adlandırılacak değişkenİyi adÖrneğiKötü adÖrneğiNeden kötü?  
Temizlenmiş Datacleaned\_datacdatit yeterince mantıklı değil.  
Clear Application Syntaxesclr\_app\_syntxix\_app\_syntaxit dizinleri yeterince anlamlı değil.  
Banka Hesaplarının Müşteri Bilgilericustomer\_bank\_infocostomer\_info\_bank\_accountçok endişeli.  
S: Aşağıdakilerden hangisi geçersiz bir ifadedir?  
A:  
A) x, y, z = 1, 22, 333  
B) x\_y\_z = 1\_234\_567  
C) xyz = 1234567  
D) x yz = 111 222 333  
  
Değişken adlarında boşluklara izin verilmez