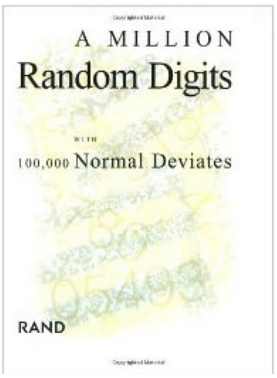
Beach reading

A Million Random Digits with 100,000 Normal Deviates is a book that was published in 1955 by the RAND Corporation. The book — that mainly consists of a table with a long sequence of randomly generated digits, as the title suggests — was important in the 20th century in the fields of statistics and random numbers. From 1947 it was constructed by means of an electronic simulation of a roulette wheel that was connected on a computer, and of which the results were meticulously filtered and tested before being added to the table.



The RAND table meant an important break-through in working with random numbers, since a table of this size and coincidence degree was not available before. Apart from the availability in book format, the digits could also be ordered on punch cards. the table was mainly used in statistics and with the experimental purpose of scientific experiments, primarily those that used so-called Monte Carlo-simulations. The book was one of the last in a sequence of tables with random numbers that were produced from the middle of the twenties until the fifties. From then on, the rise of computers allowed generating pseudo-random numbers in a faster way than they could be looked up in tables.

The book was republished in 2001 (ISBN 0-8330-3047-7) with a new prologue from Michael D. Rich, Executive Vice President of the RAND Corporation. Since then, it has received a lot of funny comments on Amazon.com:

- *"I had a hard time getting into this book. The profanity was jarring and stilted, not at all how people really talk."*
- *"Once you get about halfway in, the rest of the story is pretty predictable."*
- *"If you like this book, I highly recommend that you read it in the original binary."*
- *"I would have given it five stars, but sadly there were too many distracting typos. For example: 46453 13987."*
- *"I really liked the '10034 56429 234088' part."*
- *"Frankly the sex scenes were awkward and clumsily written, adding very little of value to the plot."*
- *"For a supposedly serious reference work the omission of an index is a major impediment. I hope this will be corrected in the next edition."*

The average readers rewards the book with four stars.

Preparation

In the random module from the Standard Python Library a datatype Random is defined, that can be

used to make pseudo-random choices. After all, a computer can never really work randomly, but always has to execute an algorithm that simulates coincidence. Such algorithms calculate, for example, a next choice based on the previous choice. The first time a choice must be made, however, these algorithms need a set start point because there was no previous choice. This is the so-called **seed**. The example session below illustrates how setting this seed can influence making random choices.

```
>>> characters = 'UNCOPYRIGHTABLE'

>>> from random import Random
>>> generator = Random()

>>> [generator.choice(characters) for _ in range(10)]
['E', 'Y', 'R', 'A', 'C', 'P', 'T', 'I', 'I', 'L']
>>> [generator.choice(characters) for _ in range(10)]
['I', 'R', 'C', 'I', 'H', 'T', 'H', 'R', 'T', 'L']

>>> generator.seed(3)
>>> [generator.choice(characters) for _ in range(10)]
['O', 'H', 'G', 'C', 'Y', 'E', 'H', 'I', 'T', 'H']
>>> [generator.choice(characters) for _ in range(10)]
['N', 'H', 'U', 'E', 'L', 'I', 'P', 'G', 'O', 'O']

>>> generator.seed(3)
>>> [generator.choice(characters) for _ in range(10)]
['O', 'H', 'G', 'C', 'Y', 'E', 'H', 'I', 'T', 'H']
>>> [generator.choice(characters) for _ in range(10)]
['N', 'H', 'U', 'E', 'L', 'I', 'P', 'G', 'O', 'O']

>>> generator.seed(17)
>>> [generator.choice(characters) for _ in range(10)]
['G', 'R', 'B', 'P', 'Y', 'P', 'C', 'B', 'A', 'A']
>>> [generator.choice(characters) for _ in range(10)]
['G', 'T', 'P', 'N', 'E', 'U', 'O', 'R', 'L', 'A']
```

In a first instance, a new object of the class `Random` is made. Because no seed is given at that moment, the system time (the time that the computer reads on its clock) is used as seed. After that, the method `choice` is used to choose a random object from a given compound object (in this case a random character from the string `UNCOPYRIGHTABLE`).

Setting a new seed is done by means of the `seed` methode. To this method, any (immutable) object can be given, that is then used as a new seed. The value `None` means that the system time will be used as seed. If you watch the session above closely, you will see that setting that setting the same seed as a start base (illustrated here by means of the seed 3) will result in the same sequence being made over and over. Because of this, you can have the randomness of you programs go predictable in a certain sense.

Assignment

Define a class `CharacterGenerator` with which a random sequence of characters can be generated in a predictable way. These characters are chosen randomly from the alphabet given. For this, the objects of the class `CharacterGenerator` must contain at least the following methods:

- An initializing method to which a seed must be given. Any (immutable) object may be given

as seed, except the value None. If the value None is given as seed, the initializing method must raise an AssertionError with the message no seed given. The initializing method must see to it that every object from the class CharacterGenerator has its own random generator (an object from the class random.Random) and that this is initialized with the given seed. The initialization method also has a second, optional parameter alphabet, to which a string of characters can be given. This forms the alphabet of the characters from which will be chosen randomly. If no explicit value was given to the parameter alphabet, there must be chosen from the digits from the alphabet 0123456789.

- A method __repr__ that prints a string representation of the object. This string representation reads as a Python expression that makes a new object from the class CharacterGenerator with the same seed and the same alphabet as the current object. Watch the example session below to find the different shapes of this string representation. Pay attention to the fact that the optional parameter must only be given if the alphabet deviates from the standard alphabet, and that, in this case, the parameter alphabet must be named explicitly.
- A method reset to which a seed can be given optionally. If a seed is given to the method, this becomes the new seed of the object. After that, the seed of the object (either the new or the old value) must be used to initialize the coincidence generator of the object.
- A method sequence that prints a string that consists of a sequence of randomly chosen characters. Here, the method choice from the coincidence generator of the object is called to select a random character from the alphabet of the object. At the end of the method sequence can be given an optional argument (standard value: 1)that indicates the length of the string that must be printed.

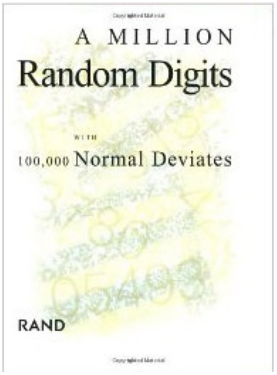
Example

```
>>> digits = CharacterGenerator(3)
>>> digits
CharacterGenerator(3)

>>> digits.sequence()
'3'
>>> digits.sequence(19)
'9825979190748337887'
>>> digits.sequence(30)
'623286012904047966697251027346'

>>> digits.sequence()
>>> digits
CharacterGenerator(3)
>>> digits.sequence(20)
'39825979190748337887'
>>> digits.sequence(30)
'623286012904047966697251027346'

>>> digits.reset(7)
>>> digits
CharacterGenerator(7)
>>> digits.sequence(20)
'52601815908301661318'
>>> digits.sequence(30)
'609139099603082462819482199351'

>>> CharacterGenerator(None)
Traceback (most recent call last):
```

AssertionError: no seed was given

```
>>> spam1 = CharacterGenerator(3, 'SPAM')
>>> spam2 = CharacterGenerator(3, 'SPAM')
>>> spam1.sequence(10)
'PPAMSSMAPP'
>>> spam2.sequence(10)
'PPAMSSMAPP'
>>> spam1
CharacterGenerator(3, alphabet='SPAM')
```

A Million Random Digits with 100,000 Normal Deviates is een boek dat in 1955 werd uitgebracht door de RAND Corporation. Het boek — dat zoals de titel reeds doet vermoeden hoofdzakelijk bestaat uit een tabel met een lange opeenvolging van willekeurig gegenereerde cijfers — was in de 20e eeuw belangrijk op het gebied van statistiek en willekeurige getallen. Het werd vanaf 1947 opgesteld door middel van een elektronische simulatie van een roulettewiel dat was aangesloten op een computer, en waarvan de resultaten vervolgens zorgvuldig gefilterd en getest werden vooraleer ze in de tabel konden opgenomen worden.



De RAND tabel betekende een belangrijke doorbraak in het werken met willekeurige getallen, aangezien een tabel van die omvang en toevalsgraad daarvoor niet voorhanden was. Naast de beschikbaarheid in boekvorm, konden de cijfers ook besteld worden op ponskaarten. De tabel werd hoofdzakelijk gebruikt in de statistiek en bij de experimentele opzet van wetenschappelijke experimenten, voornamelijk deze die gebruik maakten van zogenaamde Monte Carlo-simulaties. Het boek was één van de laatste in een reeks tabellen met willekeurige getallen die geproduceerd werden vanaf het midden van de jaren '20 tot de jaren '50. Vanaf dan liet de opkomst van computers immers toe om op een snellere manier pseudowillekeurige getallen te genereren dan dat ze in tabellen konden opgezocht worden.

Het boek werd in 2001 heruitgegeven (ISBN 0-8330-3047-7) met een nieuw voorwoord van Michael D. Rich, Executive Vice President van de RAND Corporation. Het heeft sindsdien heel wat grappige commentaren gekregen op Amazon.com:

- *"I had a hard time getting into this book. The profanity was jarring and stilted, not at all how people really talk."*
- *"Once you get about halfway in, the rest of the story is pretty predictable."*
- *"If you like this book, I highly recommend that you read it in the original binary."*
- *"I would have given it five stars, but sadly there were too many distracting typos. For example: 46453 13987."*
- *"I really liked the '10034 56429 234088' part."*
- *"Frankly the sex scenes were awkward and clumsily written, adding very little of value to the plot."*

- *"For a supposedly serious reference work the omission of an index is a major impediment. I hope this will be corrected in the next edition."*

De gemiddelde lezer geeft vier sterren aan het boek.

Voorbereiding

In de random module uit de Standard Python Library wordt een gegevenstype Random gedefinieerd, dat kan gebruikt worden om pseudo-toevallige keuzes te maken. Een computer kan immers nooit echt willekeurig werken, maar moet altijd een algoritme uitvoeren dat toeval simuleert. Dergelijke algoritmen berekenen bijvoorbeeld een volgende keuze op basis van een vorige keuze. De eerste keer dat er een keuze moet gemaakt worden, hebben deze algoritmen echter een vast vertrekpunt nodig omdat er dan nog geen vorige keuze was. Dat is de zogenaamde **seed**. Onderstaande voorbeeldsessie illustreert hoe het instellen van deze seed het maken van willekeurige keuzes kan beïnvloeden.

```
>>> karakters = 'UNCOPYRIGHTABLE'

>>> from random import Random
>>> generator = Random()

>>> [generator.choice(karakters) for _ in range(10)]
['E', 'Y', 'R', 'A', 'C', 'P', 'T', 'I', 'I', 'L']
>>> [generator.choice(karakters) for _ in range(10)]
['I', 'R', 'C', 'I', 'H', 'T', 'H', 'R', 'T', 'L']

>>> generator.seed(3)
>>> [generator.choice(karakters) for _ in range(10)]
['O', 'H', 'G', 'C', 'Y', 'E', 'H', 'I', 'T', 'H']
>>> [generator.choice(karakters) for _ in range(10)]
['N', 'H', 'U', 'E', 'L', 'I', 'P', 'G', 'O', 'O']

>>> generator.seed(3)
>>> [generator.choice(karakters) for _ in range(10)]
['O', 'H', 'G', 'C', 'Y', 'E', 'H', 'I', 'T', 'H']
>>> [generator.choice(karakters) for _ in range(10)]
['N', 'H', 'U', 'E', 'L', 'I', 'P', 'G', 'O', 'O']

>>> generator.seed(17)
>>> [generator.choice(karakters) for _ in range(10)]
['G', 'R', 'B', 'P', 'Y', 'P', 'C', 'B', 'A', 'A']
>>> [generator.choice(karakters) for _ in range(10)]
['G', 'T', 'P', 'N', 'E', 'U', 'O', 'R', 'L', 'A']
```

In eerste instantie wordt er een nieuw object van de klasse Random aangemaakt. Omdat er op dat moment nog geen seed werd opgegeven, wordt de systeemtijd (de tijd die de computer op zijn klok afleest) gebruikt als eerste seed. Daarna kan de methode choice gebruikt worden om een willekeurig element te kiezen uit een gegeven samengesteld object (in dit geval een willekeurig karakter uit de string UNCOPYRIGHTABLE).

Het instellen van een nieuwe seed gebeurt aan de hand van de seed methode. Aan deze methode kan elk (onveranderlijk) object doorgegeven worden, dat dan als nieuwe seed gebruikt wordt. De waarde None betekent dat opnieuw de systeemtijd als seed zal gebruikt worden. Als je goed kijkt naar de bovenstaande sessie, dan zal je zien dat het instellen van dezelfde seed als

vertrekbasis (hier geïllustreerd aan de hand van de seed 3) ervoor zal zorgen dat dezelfde reeks keuzes zal gemaakt worden. Hierdoor kan je ervoor zorgen dat willekeurigheid in je programma's toch enigszins voorspelbaar verloopt.

Opgave

Definieer een klasse KarakterGenerator waarmee op een voorspelbare manier willekeurige reeksen karakters kunnen gegenereerd worden. Deze karakters worden willekeurig gekozen uit een opgegeven alfabet. Hiervoor moeten de objecten van de klasse KarakterGenerator minstens over de volgende methoden beschikken:

- Een initialisatiemethode waaraan een seed moet doorgegeven worden. Als seed mag elk (onveranderlijk) object doorgegeven worden, behalve de waarde None. Indien de waarde None wordt doorgegeven als seed, dan moet de initialisatiemethode een AssertionError opwerpen met de boodschap geen seed opgegeven. De initialisatiemethode moet ervoor zorgen dat elk object van de klasse KarakterGenerator beschikt over een eigen toevalsgenerator (een object van de klasse random.Random) en dat deze geïnitialiseerd wordt met de opgegeven seed. De initialisatiemethode heeft nog een tweede optionele parameter alfabet, waaraan een string van karakters kan doorgegeven worden. Dit vormt het alfabet van de karakters waaruit willekeurig zal moeten gekozen worden. Indien er geen expliciete waarde doorgegeven wordt voor de parameter alfabet, dan moet er gekozen worden uit de cijfers van het alfabet 0123456789.
- Een methode __repr__ die een stringvoorstelling van het object teruggeeft. Deze stringvoorstelling leest als een Python-expressie die een nieuw object aanmaakt van de klasse KarakterGenerator met dezelfde seed en hetzelfde alfabet als het huidige object. Bekijk onderstaande voorbeeldsessie om de verschillende vormen van deze stringvoorstelling te achterhalen. Let hierbij onder andere op het feit dat de optionele parameter enkel moet opgegeven worden indien het alfabet afwijkt van het standaardalfabet, en dat de parameter alfabet in dat geval expliciet moet benoemd worden.
- Een methode reset waaraan optioneel een seed kan doorgegeven worden. Indien een seed wordt doorgegeven aan de methode, dan wordt dit de nieuwe seed van het object. Daarna moet de seed van het object (hetzij de nieuwe, hetzij de oude waarde) gebruikt worden om de toevalsgenerator van het object opnieuw te initialiseren.
- Een methode reeks die een string teruggeeft bestaande uit een reeks willekeurig gekozen karakters. Hierbij moet de methode choice van de toevalsgenerator van het object aangeroepen worden, om telkens een willekeurig karakter te selecteren uit het alfabet van het object. Aan de methode reeks kan optioneel een argument doorgegeven worden (standaarwaarde: 1) dat de lengte van de string aangeeft die moet teruggegeven worden.

Voorbeeld

```
>>> cijfers = KarakterGenerator(3)
>>> cijfers
KarakterGenerator(3)

>>> cijfers.reeks()
'3'
>>> cijfers.reeks(19)
'9825979190748337887'
>>> cijfers.reeks(30)
'623286012904047966697251027346'
```

```
>>> cijfers.reset()
>>> cijfers
KarakterGenerator(3)
>>> cijfers.reeks(20)
'39825979190748337887'
>>> cijfers.reeks(30)
'623286012904047966697251027346'

>>> cijfers.reset(7)
>>> cijfers
KarakterGenerator(7)
>>> cijfers.reeks(20)
'52601815908301661318'
>>> cijfers.reeks(30)
'609139099603082462819482199351'

>>> KarakterGenerator(None)
Traceback (most recent call last):
AssertionError: geen seed opgegeven

>>> spam1 = KarakterGenerator(3, 'SPAM')
>>> spam2 = KarakterGenerator(3, 'SPAM')
>>> spam1.reeks(10)
'PPAMSSMAPP'
>>> spam2.reeks(10)
'PPAMSSMAPP'
>>> spam1
KarakterGenerator(3, alfabet='SPAM')
```