Curso Python Avançado

Classes e Iteradores

olist

Classes

Classes

Detalhes Sobre Contexto

Código disponível em: https://pastebin.com/uwB4gbuL

```
kind = "blank"

def test_kind():
    print("Test Kind:", kind)

class Dog:
    kind = 'canine'

    def __init__(self, name):
        self.name = name

    def what_kind(self):
        print("Name:", self.name)
        print("Geral:", kind)
        print("Self:", self.kind)
```

```
# Testes...
d = Dog('Fido')
e = Dog('Buddy')

test_kind()
print('-' * 40)

d.what_kind()
print('-' * 40)
e.what_kind()
print('-' * 40)

print(Dog)
print(Dog.what_kind)
print(e)
print(e.what_kind)
```

Classes

Invocando Métodos

Métodos devem sempre ser invocados utilizano "self".

```
class Bag:
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def addtwice(self, x):
        self.add(x)
        self.add(x)
```

Classes

Herança

A sintaxe para definir uma classe que herda de outra é:

```
class DerivedClassName(BaseClassName):
```

Óbviamente, também é possível herdar de classes definidas em outros módulos:

```
class DerivedClassName(modname.BaseClassName):
```

Classes

Herança

Para chamar um método de uma classe pai:

```
>>> BaseClassName.methodname(self, arguments)
```

Outra maneira mais pratica é usando o builtin `super()`:

```
>>> super().methodname(arguments)
```

Classes

Herança

Python também suporta herança múltipla:

```
class DerivedClassName(Base1, Base2, Base3):
```

Classes

Atributos Privados

Atributos privados devem sempre começar com dois caracteres _underline_ "__private_attribute".

```
class Mapping:
    def __init__(self, iterable):
        self.items_list = []
        self.__update(iterable)

    def update(self, iterable):
        for item in iterable:
            self.items_list.append(item)
```

Iteradores

Iteradores

Convenção

Objetos iteráveis nada mais são do que uma convenção de chamada implícita dos métodos `__iter__()` e `__next__()`.

```
>>> s = 'abc'
>>> it = iter(s)
>>> it
<iterator object at 0x00A1DB50>
>>> next(it)
'a'
>>> next(it)
'b'
>>> next(it)
'c'
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    next(it)
StopIteration
```

Iteradores

Criando um Iterador

Para criar uma classe iteravel, declare os métodos __iter__ e __next__.

```
class Reverse:
    """Iterator for looping over a sequence backwards."""
    def __init__(self, data):
        self.data = data
        self.index = len(data)

    def __iter__(self):
        return self

    def __next__(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]
```

Geradores

Geradores

Geradores são uma forma de gerar dados iteraveis de maneira compacta, fazendo uso de “yield”

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]
```