Sets

Características

Sets são um tipo de estrutura de dados não ordenada e não mutável, o que significa que não podemos nem acessar seus elementos a partir do indice, nem realizar alterações nos valores que compõem os sets, apesar disso é possível adicionar ou remover itens de um set. Apesar dessas características podemos utilizar a estrutura de repetição for para percorrer os elementos de um set e realizar operações com eles.

Outra característica do set, é que valores duplicados não são permitidos, se por algum motivo for declarado um set com dois valores idênticos, apenas a primeira ocorrência deste valor será considerada. Para declarar um set é necessário usar chaves, vamos ver um exemplo:

```
my_set = {1, 2, 3}
other_set = {"Antônio", 4, 78.4321}
```

Simples certo? A principal diferença em relação as listas e tuplas consiste no fato da coleção ser desordenada, e, portanto, não existem índices, o que torna muito complicado acessar um item específico dentro do set, mas podemos utilizar um for para printar cada um dos itens do set:

```
other_set = {"Antônio", 4, 78.4321}
for item in other_set:
        print(item)
```

Se executarmos um arquivo python com o conteúdo do exemplo acima várias vezes, vamos observar algo de muito curioso, veja os resultados de três tentativas realizadas por nós:

```
>> Antônio
>> 4
>> 78.4321

>> 4
>> 78.4321
>> Antônio

>> 78.4321
>> 4
>> Antônio
```

Isso acontece porque a estrutura não é ordenada em memória, dessa forma, cada vez que o script python é executado, os valores são inseridos na memória do computador de maneira aleatória.

Existem outras curiosidades relacionadas aos sets, um exemplo é a criação de um set vazio, como os sets compartilham o uso das chaves com os dicionários (uma outra estrutura de dados em python), não podemos criar um set sem valores apenas abrindo e fechando chaves, para isso, podemos utilizar a função construtora set().

```
# Dicionário
not_a_set = {}

# Set vazio
empty_set = set()
```

Modificando sets

Para adicionar um único valor dentro de um set, podemos usar o método add(valor):

```
my_set.add(1)
```

Se for necessário adicionar mais de um valor ao mesmo tempo, o método update(lista), pode adicionar uma lista, uma tupla ou até mesmo um outro set e adicionar todos os itens dessa coleção dentro do set especificado, lembrando que caso existam itens duplicados, estes não serão inseridos.

```
my_set.update([1, 2, 3])
my_set.update((4, 5, 6))
```

Removendo valores

Apesar dos sets não serem indexados, é possível remover valores de um set utilizando os métodos discard() e remove(). Ambos realizam a remoção a partir do valor informado, o que não é um problema neste caso, já que não podemos inserir valores duplicados dentro de um set, a única diferença entre os dois métodos só é observada quando o valor a ser removido não se encontra no set, que no caso do discard() nada acontece, mas se tentarmos remover um valor não presente no set utilizando remove(), vamos receber um erro.

```
my_set = {1, 2, 3}
my_set.discard(1)
print(my_set) # {2, 3}

my_set.remove(2)
print(my_set) # {3}

my_set.discard(1) # nada acontece
my_set.remove(1) # KeyError
```

Por fim, é possível limpar (remover todos os valores) de um set utilizando o método clear().

```
my_set = {1, 2, 3}
my_set.clear()
print(my_set) # {}
```

Operações envolvendo sets

Os sets possuem uma gama gigantesca de métodos que podem ser utilizados para realizar diversas operações envolvendo um ou mais sets, mas talvez a principal funcionalidade dos sets são algumas operações matemáticas específicas, como a união, intersecção, diferença e diferença simétrica, vamos conhecer formas de calcular cada uma dessas.

União

Podemos unir dois sets diferentes (ignorando valores repetidos) utilizando o operador de união "|" ou o método union(), vamos ver alguns exemplos de código, os resultados estão comentados:

```
set_a = {1, 2, 3}
set_b = {3, 4, 5}

# Operador |
set_c = set_a | set_b
print(set_c) # {1, 2, 3, 4, 5}

# Método union()
set_c = set_a.union(set_b)
print(set_c) # {1, 2, 3, 4, 5}
```

Intersecção

A intersecção de sets consiste em selecionar os valores que são comuns nos sets da expressão, valores que não estejam presentes nos dois sets, são descartados. O operador da intersecção é o "&"

```
set_a = {1, 2, 3}
set_b = {3, 4, 5}

# Operador &
set_c = set_a & set_b
print(set_c) # {3}

# Método intersection()
set_c = set_a.intersection(set_b)
print(set_c) # {3}
```

Diferença

Existem dois tipos de diferenças matemáticas, a diferença de set, que verifica os valores que estão contidos no set em questão, mas não no set que está sendo comparado, e a diferença simétrica, que será abordada mais adiante. A diferença de set, verifica quais itens do set A não estão presentes no set B, e retorna estes itens. O operador da diferença é o sinal de menor "-"

```
set_a = {1, 2, 3}
set_b = {3, 4, 5}

# Operador -
set_c = set_a - set_b
print(set_c) # {1, 2}

# Método difference()
set_c = set_a.difference(set_b)
print(set_c) # {1, 2}
```

Diferença simétrica

Em contraste a diferença de sets, a diferença simétrica considera valores que não sejam comuns aos dois sets comparados, ou seja, os valores que existem no set A, que não existem também no set B, retornando também os valores únicos do set B. O operador de diferença simétrica é o ^, mas podemos utilizar também o método `symmetric_difference()`.

```
set_a = {1, 2, 3}
set_b = {3, 4, 5}

# Operador ^
set_c = set_a ^ set_b
print(set_c) # {1, 2, 4, 5}

# Método difference()
set_c = set_a.symmetric_difference(set_b)
print(set_c) # {1, 2, 4, 5}
```

Sets são estruturas de dados bastante complexas e ao mesmo tempo muito úteis pela alta gama de operações disponíveis na linguagem python, nesta aula não abordamos todas as possibilidades de operações com **sets**, mas já é o suficiente para dar continuidade no processo de aprendizado da linguagem.