Curso Python Avançado

Estruturas e Funções Variádicas

olist

Tuplas e Dicionários

Tuplas e Dicionários

Tuplas

Tupla é um objeto do tipo sequencial, imutável, geralmente utilizado para armazenamento de dados heterogêneos (tipos diferentes).

```
>>> a = ()
>>> b = 1,
>>> c = (1,)
>>> d = 0, "Texto", 2
>>> e = (0, "Texto", 2)
>>> f = tuple()
```

Tuplas e Dicionários

Tuplas

Tupla também pode ser criada a partir de qualquer outro objeto iterável.

```
>>> tuple([0, 1, 2])
>>> tuple("abc")
```

Tuplas e Dicionários

Dicionários

Dicionário é um objeto de mapeamento, mutável, que utiliza objetos *hashable* como chave de mapeamento para objetos arbitrários e heterogêneos.

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict([('two', 2), ('one', 1), ('three', 3)])
>>> d = dict({'three': 3, 'one': 1, 'two': 2})
>>>
>>> a == b == c == d
```

Tuplas e Dicionários

Dicionários

Dicionários devem sempre ser construídos como uma lista de pares...

```
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
```

Funções Variádicas

Funções Variádicas

Argumentos Arbitrários

Funções podem ser definidas de modo a aceitar um número arbitrário de argumentos.

```
def func_variadic(param, *args):
    print("param:", param)
    for arg in args:
        print(arg)

func_variadic("test", "More", "and", "more", "arguments")
```

Funções Variádicas

Argumentos Arbitrários

Em geral, argumentos variádicos são os últimos na lista de parâmetros. Qualquer argumento informado após parâmetros variádicos podem ser informados apenas com *keywords*.

```
def concat(*args, sep="/"):
    return sep.join(args)

concat("earth", "mars", "venus")
concat("earth", "mars", "venus", sep=".")
```

Funções Variádicas

Argumentos Arbitrários por Keyword

Parâmetros definidos com o formato `**parametro` recebem um dicionário com todas as definições por *keywords* passadas como argumento na chamada da função.

```
def func_var_keyword(param, **keywords):
    print("param:", param)
    for kw in keywords:
        print(kw, ":", keywords[kw])

func_var_keyword("test",
           keyword_a="Value A",
           keyword_b="Value B",
           keyword_c="Value C")
```

Funções Variádicas

Argumentos Arbitrários por Keyword

Ambos os métodos de definição e invocação de funções variádicas podem ser usadas em conjunto.

```
def func_variadic_full(param, *args, **keywords):
    print("param:", param)
    print("-" * 40)
    for arg in args:
        print(arg)
    print("-" * 40)
    for kw in keywords:
        print(kw, ":", keywords[kw])

func_variadic_full("test",
           "More", "and", "more", "arguments",
           keyword_a="Value A",
           keyword_b="Value B",
           keyword_c="Value C")
```

Funções Variádicas

Desempacotando Listas de Argumentos

Argumentos podem ser passados para funções a partir de listas, tuplas ou dicionários.

Uma lista de argumentos pode ser passado utilizando o operador `*`.

```
>>> list(range(3, 6))
>>> args = [3, 6]
>>> list(range(*args))
```

Funções Variádicas

Desempacotando Listas de Argumentos

Argumentos identificados por *keywords* podem ser passador utilizando o operador **.

```
>>> def add(a=0, b=0):
...     return a + b

>>> d = {'a': 2, 'b': 3}
>>> add(**d)
```

PEP 8

PEP 8

PEP 8 tornou-se o guia de estilo adotado pela maioria dos projetos; promove um estilo de codificação muito legível e visualmente agradável.

Principais pontos:

- Use indentação com 4 espaços, e não use tabulações.
- Quebre as linhas de modo que não excedam 79 caracteres.
- Deixe linhas em branco para separar funções e classes, e blocos de código dentro de funções.
- Quando possível, coloque comentários em uma linha própria.
- Escreva strings de documentação.
- Use espaços ao redor de operadores e após vírgulas, mas não diretamente dentro de parênteses, colchetes e chaves: a = f(1, 2) + g(3, 4).
- Nomeie suas classes e funções de forma consistente; a convenção é usar MaiusculoCamelCase para classes e minusculo_com_underscores para funções e métodos. Sempre use self como o nome para o primeiro argumento dos métodos (veja Primeiro contato com classes para mais informações sobre classes e métodos).
- Não use codificações exóticas se o seu código é feito para ser usado em um contexto internacional. O padrão do Python, UTF-8, ou mesmo ASCII puro funciona bem em qualquer caso.
- Da mesma forma, não use caracteres não-ASCII em identificadores se houver apenas a menor chance de pessoas falando um idioma diferente ler ou manter o código.

PEP 8

Documentação disponível em: https://www.python.org/dev/peps/pep-0008/
Conteúdo para estudo em: https://realpython.com/python-pep8/