Revisão

Variáveis

```
>>> v = 1
>>> type(v)
<type 'int'>
>>> v = 4.
>>> type(v)
<type 'float'>
>>> v = "Boa tarde"
>>> type(v)
<type 'str'>
>>> v = []
>>> type(v)
<type 'list'>
>>> v = ()
>>> type(v)
<type 'tuple'>
>>> v = {}
>>> type(v)
<type 'dict'>
>>> v = set()
>>> type(v)
<class 'set'>
```

Operadores lógicos

```
>>> a = 1
>>> b = 2
>>> a == b # igualdade
False
>>> a!= b # diferença
True
>>> a > b # maior que
False
>>> a < b # menor que
True
>>> 2*a > b #
False
>>> 2**3 # exponenciação
8
>>> 2**(3+6)
512
>>> 7 % 2 # resto da divisão
1
```

Listas

São elementos entre colchetes separados por vírgula.

Operações com listas

```
lst = [] # lista vazia
lst[i] = 3 # substitui um elemento
lst[i:j] = 5 #substitui um grupo de elementos
del [i:j] # remove um grupo de elementos
lst.append("a") # adiciona um elemento
lst.extend([1,2,3]) # adiciona uma lista
lst.index(3) # retorna o número índice do elemento
lst.insert(i,x) # insere um elemento na posição x
lst.pop() # retire o último elemento
lst.remove(x) # um elemento
lst.reserve() # organiza a lista em ordem decrescente
lst.sort() # organiza a lista em ordem crescente
```

Tuplas

Tuplas são como listas, mas, imutáveis, assim como ocorre com strings, ou seja, não podemos acrescentar, apagar ou fazer atribuições aos itens.

```
>>> tpl = (a, b, …, z)
>>> #Obs. Os parênteses são opcionais.
>>> #Uma tupla com apenas um elemento é representada assim:
>>> tpl = (1,)
>>> primeiro_elemento = tupla[0]
>>> #Convertendo uma lista em uma tupla:
>>> tpl = tuple(lista)
>>> #Convertendo uma tupla em uma lista:
>>> lst = list(tpl)
```

Embora uma tupla possa conter elementos mutáveis, esses elementos não podem sofrer atribuição, pois isto modificaria a referência ao objeto.

```
>>> tpl = ([1, 2], 4)
>>> tpl[0].append(3)
>>> tpl
([1, 2, 3], 4)
>>> tpl[0] = [1, 3, 4]
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    tpl[0] = [1,3,4]
TypeError: 'tuple' object does not support item assignment
```

As tuplas são mais eficientes do que as listas convencionais, pois consomem menos recursos computacionais (memória), por serem estruturas mais simples.

Dicionários

São um conjunto de objetos (chave: valor), separados por vírgula entre chaves {}.

```
>>> {'andre': 46,}
```

Operações com dicionários

```
>>> a = {'nome':'André', 'idade': 39}
>>> len(a) # número de elementos
>>> a['nome'] # valor de nome
>>> a['nome'] = 'Pedro' # atribui um valor (Pedro) para chave "nome"
>>> del(a['nome']) # deleta a chave e seu valor
>>> a.clear() # apaga todos os elementos
>>> dct = a.copy() # retorna uma cópia do dicionário
>>> a.items() # retorna uma lista de tuplas (chave,valor) de todos os elementos
>>> a.keys() # retorna uma lista de todas as chaves
>>> a.values() # retorna uma lista de todos os valores
```

Conjunto

Set é uma collection que representa conjunto.

Essa coleção tem como característica ser desordenada e ter os elementos únicos, ou seja, não existe repetição de elementos nesse tipo de collection.

Existem também métodos que fazem operações de conjunto, como por exemplo union e intersection que respectivamente retornam todos os elementos de dois set e os elementos que eles tem em comum.

```
>>> a = {1, 2, 3, 4}
>>> b = {3, 4, 5, 6}
>>> print(a.union(b))
{1, 2, 3, 4, 5, 6}
>>> l1 = [1, 2, 3]
>>> l2 = [2, 4, 3]
>>> print(set(l1).intersection(l2))
{2, 3}
```

Operação	Equivale	Resultado
len(s)		número de elementos no conjunto *s* (cardinalidade)
x in s		teste x para associação em *s*
x not in s		teste x para não associação em *s*
s.issubset(t)	s <= t	testar se cada elemento em *s* está em *t*
s.issuperset(t)	s >= t	testar se cada elemento em *t* está em *s*
s.union(t)	s | t	novo conjunto com elementos de *s* e *t*
s.intersection(t)	s & t	novo conjunto com elementos comuns a *s* e *t*
s.difference(t)	s - t	novo conjunto com elementos em *s*, mas não em *t*
s.symmetric_difference(t)	s ^ t	novo conjunto com elementos em *s* ou *t*, mas não em ambos
s.copy()		novo conjunto com uma cópia baixa de *s*
s.add(x)		adicione o elemento x para definir s
s.remove(x)		remova *x* do conjunto *s*; levanta KeyError se não estiver presente
s.discard(x)		remove *x* do conjunto *s* se presente
s.pop()		remove e retorna um elemento arbitrário de *s*; levanta *KeyError* se vazio
s.clear()		remova todos os elementos do conjunto *s*

Estrutura de controle

Teste (if...elif...else...)

É uma estrutura que executa determinados blocos de comandos na dependência de sua condição (expressão) ser verdadeira.

Sintaxe:

if (expressão):

```
     Bloco comando
[elif (expressão):
     Bloco de comando
else:
     Bloco de comando]
```

```
>>> a = 1
>>> if a == 1:
      print("É o número um")
else:
      print("não é o número um")
```

Loop (while...)

É uma estrutura de repetição que executa um bloco de comando enquanto a condição (expressão) for verdadeira.

Sintaxe:

```
while (expressão):
     Bloco de comando
[else:
     Bloco de comando]
```

```
a = 0
while a < 10:
      print(a)
      a = a + 1 # a += 1
```

Varredura (for...)

É uma estrutura de varredura de objetos sequenciáveis (range(), string, lista, tupla, dicionário etc.).

Sintaxe:

```
for variável in sequencia:
     Bloco de comando
[else:
     Bloco de comando]
```

```
for i in range(3):
      print(i)
```

Controle de fluxo (pass, break, continue)

pass – define um bloco de comando vazio.

```
>>> z = 0
>>> if z == 1:
      pass
```

break – termina o loop imediatamente, usado somente nas estruturas while e for.

```
>>> a = 0
>>> while a < 10:
      a += 1
      if a == 5:
         break
      print(a)
```

continue – volta ao início do bloco de comando e continua a próxima interação, usado somente nas estruturas while e for.

```
>>> a = 0
>>> b = 0
>>> while a < 10:
      a += 1
      if b >= 5:
         continue
      b += 1
    print(a, b)
```

Exercícios

1. Dados três valores: A, B, C. Verificar se eles podem ser classificados como triângulo: equilátero, isósceles ou qualquer. Os valores são inteiros e positivos.
2. Construir um algoritmo para calcular a média de um conjunto de valores inteiros e positivos.