Fundamentos de algoritmos e introdução à programação em python

Prof. Everson Otoni

Variáveis, expressões e instruções

Um dos recursos eficientes de uma linguagem de programação é a capacidade de manipular variáveis.

Uma variável é um nome que se refere a um valor.

Instruções de atribuição

Uma *instrução de atribuição* cria uma nova variável e dá um valor a ela:

```
>>> mensagem = 'Esta é uma mensagem de teste'
>>> numero = 17
>>> pi = 3.141592653589793
```

Existem aqui três atribuições.

A primeira atribui uma string a uma nova variável chamada mensagem.

A segunda dá o número inteiro 17 a variável chamada numero.

A terceira atribui o valor (aproximado) de π a variável denominada de "pi".

PEP 8

PEP

Documento que fornece convenções de codificação para o código Python que compreende a biblioteca padrão na distribuição principal do Python.

A PEP 8

Escrito pelo próprio fundador do Python, o Guido van Rossum.

Guia de estilo que evolui com tempo à medida que convenções adicionais são identificadas ou se tornem obsoletas devido a mudanças na própria linguagem.

É um conjunto de diretrizes que têm como objetivo melhorar a legibilidade do código e torná-lo consistente em todo o amplo espectro do código Python.

PEP 8 - Convenções de nomenclatura

Essas convenções ainda são confusas, mas existem certos padrões de nomenclatura que atualmente são recomendados.

Princípio primordial

Os nomes que são visíveis para o usuário devem seguir convenções que reflitam o uso e não a implementação.

PEP 8 - Convenções de nomenclatura

Estilos de nomenclatura

Os seguintes estilos de nomenclatura são comumente distinguidos em:

- b (única letra minúscula)
- B (única letra MAIÚSCULA)
- minúsculas
- minúsculas_separadas_por_sublinhados
- MAIÚSCULAS
- MAIÚSCULAS_SEPARADAS_POR_SUBLINHADOS

PEP 8 - Convenções de nomenclatura

Estilos de nomenclatura

- PalavrasComecamPorMaiusculas (Corcova do camelo - CamelCase ou CapWords)
- palavrasComecamPorMaisculasComExcecaoDaPrimeiraPalavra (Variação da "Corcova do camelo")

PEP 8 - Convenções de nomenclatura

Nomes a evitar

Nunca use os caracteres 'l' (L minúsculo), 'O' (o maiúsculo) ou 'I' (i maiúsculo) sozinhos como nomes de variáveis. Em algumas fontes, esses caracteres são indistinguíveis dos números um e zero.

Nome das variáveis

Geralmente e recomenda-se escolher nomes significativos para as variáveis.

Nomes de variáveis podem ser tão longos quanto você queira

Podem conter letras com números, mas nunca podem começar com um número.

A convenção é utilizar apenas letras minúsculas para nomes de variáveis.

Nome das variáveis - Nomes que revelem o seu propósito

Escolher bons nomes leva tempo, mas economiza mais.

O nome de uma variável, função ou classe deve:

- **Lhe dizer porque existe**
- **O que faz**
- **Como é usado**

Dica: Se um nome requer um comentário de código, então não revela seu propósito.

Nome das variáveis - Nomes que revelem o seu propósito

```
d = 25 # dias decorridos do pagamento
```

O nome "d" não revela nada. Não indica a ideia de tempo decorrido, nem de dias e nem com o que o tempo está relacionado.

```
dias_decoridos_pagamento = 25
```

Nome das variáveis - Faça distinções significativas

Os programadores criam problemas para si mesmos quando criam um código voltado unicamente para o compilador ou interpretador.

Se os nomes precisam ser diferentes, então também devem ter significados distintos.

Não basta adicionar números ou palavras comuns, mesmo que o compilador ou interpretador fique satisfeito.

Nome das variáveis - Faça distinções significativas

Por exemplo:

```
# Errado
a1 = 'Bernado'
a2 = 'Beatriz'
a3 = 'Caio'
```

Usar números sequenciais em nomes (a1, a2, a3,...,aN) é o oposto de nome expressivos.

Geram confusão, simplesmente não oferecem informação alguma ou dica sobre a intenção de seu criador.

```
# Certo
nome_dos_alunos = ['Bernardo', 'Beatriz', 'Caio']
```

Nome das variáveis - Faça distinções significativas

Palavras muito comuns são outra forma de distinção que nada expressam.

Palavras muito comuns são redundantes. O nome de uma variável jamais deve conter a palavra "variável".

Faça a distinção dos nomes de uma forma que o leitor compreenda as diferenças

Nome das variáveis - Use nomes pronunciáveis

Use a parte do cérebro que evoluiu para lidar com a língua falada para nomes pronunciáveis.

A programação é uma atividade social

```
# Errado
gedtamdhms()
```

No caso acima a função gedtamdhms (gerador de data, ano, mês, dia, hora, minuto e segundo). Se pronunciaria como "gê dê tê a eme dê agá eme ése"

```
# Certo
gerador_data()
```

Nome de variáveis - Use nomes passíveis de busca

Nomes de uma só letra ou números não são fáceis de localizá-los ao longo de um texto.

Por exemplo, pode-se localizar facilmente a variável *quantidade_max_materias_por_estudante*, do que localizar o número *7*.

Nomes, definições e várias outras expressões que possuem tal número, usado para outros propósitos podem ser resultados da busca.

Logo, nomes longos se sobressaem aos curtos.

Nome de varáveis - Evite o mapeamento mental

Os leitores não devem ter de traduzir mentalmente os nomes que você escolheu por outros que eles conheça.

Não use termos do domínio do problema e nem os da solução.

Esse é o problema com nomes de variáveis de uma só letra.
Certamente um contador de iterações pode ser chamado de "i", "j" ou "k" - isso já se tornou uma tradição.

Nomes de varáveis - Não dê uma de espertinho

Se os nomes forem muito "espertinhos", apenas as pessoas que compartilhem do mesmo senso de humor que seu dono irão entendê-los ou lembrá-los.

```
# Certo
granada()
```

Saberão o que a função *granada* faz? Talvez. É engraçado, mas talvez seja melhor uma função chamada de *apaga_itens*.

```
# Certo
apaga_itens()
```

Essas brincadeiras em código costumam aparecer na forma de coloquialismos e gírias. E até mesmo piadas de baixo calão.

Diga o que você quer expressar, expresse o que você quer dizer.

Nomes de varáveis - Adicione um contexto significativo

Há poucos nomes que são significativos por si só - a maioria não é.

Utilize nomes que façam parte do contexto do leitor. E em último recurso adicione prefixos ao nome.

Imagine as varáveis chamadas *primeiro_nome*, *sobrenome*, *logradouro*, *numero*, *cidade*, *estado*, *cep*. Vistas juntas, fica bem claro que ela formam um endereço.

Mas e se fosse visto as variáveis *numero* e *estado* sozinhas? Seria assumido que fazem parte de um endereço?

Expressões

Uma expressão é uma combinação de valores, variáveis e operadores.

Um valor por si mesmo é considerado uma expressão, assim como uma variável, portanto as expressões seguintes são todas legais.

```
>>> 42
42
```

```
>>> numero_de_dias
17
```

```
>>> numero_de_dias + 10
27
```

Quando se digita uma expressão no prompt, o interpretador a avalia, ou seja, ele encontra o valor da expressão. No exemplo, o *numero_de_dias* tem o valor *17* e *numero_de_dias + 10* tem o valor *27*.

Instrução

Uma instrução é uma unidade de código que tem um efeito, como criar uma variável ou exibir um valor.

Uma instrução de atribuição:

```
>>> nota_de_matematica = 9.5
```

Uma instrução de exibição:

```
>>> print(nota_de_matematica)
```

Quando se digita uma instrução, o interpretador a executa, o que significa que ele faz exatamente o que a instrução diz.

Operações com strings

Em geral, não é possível executar operações matemáticas com strings, mesmo se elas parecerem números.

```
# Errado
'2' - '1'
'python' / 'programação'
'leitura' * 'prática'
```

Há duas exceções, *+* e ***

Operações com strings

O operador *+* executa uma concatenação de strings, ou seja, une as strings pelas extremidades. Por exemplo

```
>>> nome_usuario = 'brenno'
>>> servidor_email = '@gmail.com'
>>> print(nome_usuario + servidor_email)
brenno@gmail.com
```

Operações com strings

O operador `*` também funciona em strings; ele executa a repetição.

```
>>> nome_usuario = 'brenno'
>>> print(nome_usuario * 3)
brennobrennobrenno
```

Se um dos valores for uma string, o outro tem de ser um número inteiro.

O uso de `+` e `*` faz sentido por analogia com a adição e multiplicação.

Comentários

Muitas vezes mesmo que faça o uso de bons nomes para variáveis, funções, classes entre outras atribuições. Haverá a necessidade de acrescentar notas aos programas desenvolvidos.

Essas notas são chamadas de comentários.

```
# Registra a porcentagem dos minutos que passaram

porcetagem_minutos = (minutos_decorridos / 60) * 100
```

Nesse caso, o comentário aparece unicamente em uma linha.

Comentários

Também é possível pôr comentários no fim das linhas:

```
porcetagem_minutos = (minutos_decorridos / 60) * 100 # Registra a porcentagem dos minutos que passaram
```

Tudo do # ao fim da linha é ignorado - não tem efeito na execução do programa.

Bons nomes de variáveis, como discutimos, reduzem a necessidade de comentários.

Nomes longos podem tornar expressões complexas de ler, nessas horas os comentários serão muito bem-vindos.

Depuração

Existem três tipos de erros que podem ocorrer em um programa.

- Erros de sintaxe
- Erros de tempo de execução
- Erros semânticos

Depuração

Erros de sintaxe

A "sintaxe" refere-se à estrutura de um programa e suas respectivas regras.

Por exemplo, estruturalmente os parênteses devem vir em pares correspondentes.

```
# Errado
1 + 2)
File "<stdin>", line 1
    1 + 2)
          ^
SyntaxError: unmatched ')'
```

No inicio você pode passar muito tempo rastreando erros de sintaxe, ao adquirir experiência, você fará menos erros e os encontrará mais rápido.

Depuração

Erros de tempo de execução

Esse tipo de erro não aparece até que o programa seja executado. Esses erros também se chamam de exceções porque normalmente indicam que algo excepcional (e ruim) aconteceu.

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Depuração

Erros semânticos

O terceiro tipo de erro é "semântico", ou seja, relacionado ao significado.

Se houver um erro semântico no seu programa, ele será executado sem gerar mensagens de erro, mas não vai fazer a coisa certa. Vai fazer algo diferente, mas especificamente, vai fazer o que você disse para fazer.

Identificar erros semânticos pode ser complicado, porque é preciso trabalhar de trás para a frente, vendo a saída do programa e tentando compreender o que ele está fazendo.