Variáveis, Expressões e Comandos

No início do aprendizado de qualquer linguagem de programação existem alguns conceitos e idéias básicas que são necessários. O objetivo deste capítulo é introduzi-lo ao vocabulário básico de programação e alguns dos conceitos fundamentais de Python.

Variáveis e tipos de dados

Um **valor** é um dos coisas fundamentais — como uma palavra ou número — que um programa manipula. Os valores que vimos até o momento são `5` (o resultado quando fazemos a adição `2 + 3`), e `"Olá, mundo!"`. Nós frequentemente nos referimos a esses valores como **objetos** e usaremos as palavras valor e objeto indiscriminadamente.

Note

Na verdade, o 2 e o 3 que são parte da adição acima são também valores (objetos).

Esses objetos são classificados em **classes** ou **tipos de dados** diferentes: `4` é um *inteiro*, e `"Olá, mundo!"` é um *string* ou *texto cadeia de caracteres*, que recebe esse nome pois contém uma sequência de letras ou caracteres. (N.T. Utilizamos também o termo em inglês *string* já que esse é comumente usado por programadores.) Você (e o interpretador) podem identificar strings pois estes estão envolvidos por aspas.

Se você não está seguro sobre a classe a que pertence um valor, Python tem uma função chamada **type** que pode dizer-lhe isto.

```
1 print(type("Ola, mundo!"))
2 print(type(17))
3 print("Ola, mundo")
4
5
```

ActiveCode: 1 (ch02_1)

Run Save Load

Não é surpresa que strings pertençam a uma classe chamada **str** e inteiros pertencem a classe chamada **int**.

Note

Quando mostramos o valor de um string usando a função `print`, como na terceira linha acima, as aspas não estão presentes. O valor de um

string é a sequência de caracteres entre as aspas. As aspas são apenas necessárias para ajudar o Python a saber qual é o valor delimitando-o.

No Python shell, não é necessário usar a função `print` para ver o valor mostrado acima. O shell executa a função e automaticamente imprime o resultado. Por exemplo, consider a seção do shell exibida abaixo. Quando pedimos para o shell executar `type("Hello, World!")`, ele imprime a reposta apropriada e na próxima linha continua exibindo o prompt para um novo uso.

```
Python 3.3.0 (default, Sep 29 2012, 17:14:58)
[GCC 4.7.2] on linux
Type "copyright", "credits" or "license()" for more information.
>>> type("Ola, mundo!")
<class 'str'>
>>> type(17)
<class 'int'>
>>> "Ola, mundo!"
'Ola, mundo!'
```

Observe que no último exemplo, nós simplesmente pedimos ao shell que avaliasse o string "Ola, mundo". O resultado é como você poderia imaginar, o próprio string.

Continuando com nossa discussão sobre tipos de dados, números com ponto decimal (e não vírgula decimal) pertencem à classe chamada **float**, pois esses números são representados em uma forma que é chamada de *ponto flutuante* (*floating-point*). No presente estágio, você pode usar as palavras *classe* e *tipo* indiscriminadamente. Em capítulos mais adiante nós voltaremos a tratar de classe e a buscar uma compreensão mais profunda desse conceito.

```
1 print(type(3.2))
2
3
```

ActiveCode: 2 (ch02_2)

Run Save Load

E valores como `"17"` e `"3.2"`? Eles parecem números, mas eles estão envolvoltos entre aspas como um string.

```
1 print(type("17"))
2 print(type("3.2"))
3
4
```

ActiveCode: 3 (ch02_3)

Run Save Load

Eles são strings!

Strings em Python podem ser delimitados por apóstrofos (`'`) aspas (`"`), ou três de cada (`'''` ou `"""`)

```
1 print(type('Esse e um string.') )
2 print(type("E esse tambem eh um string.") )
3 print(type("""e esse.""") )
4 print(type('''e mesmo esse...''') )
5
6
```

ActiveCode: 4 (ch02_4)

Run Save Load

Strings com aspas podem conter apóstrofos, como em `"O símbolo ' é um apóstrofo"`, e strings com apóstrofos podem conter aspas, como em `'Os cavaleiros que dizem "Ni!"'`. Strings delimitados por três aspas ou apóstrofos são chamadas de strings triplos (*triple quoted strings*). Eles podem conter aspas, apóstrofos ou strings entre aspas ou apóstrofos:

```
print('''"Oh nao", exclamou ela, "A bicleta esta
```

ActiveCode: 5 (ch02_5)

Run Save Load

Strings triplos podem até se estender por várias linhas:

```
mensagem = """Esta mensagem ira
se estende varias
linhas."""
print(mensagem)

print("""Esta mensagem se estende
por varias linhas
do texto.""")
```

ActiveCode: 6 (ch02_6)

Run Save Load

Python não se importa se você usa aspas, apóstrofes, aspas triplas ou apóstrofes triplos para envolver um string. Uma vez verificado que o texto do seu programa ou comando está sintaticamente correto, a maneira com que o valor será armazenada é identica em todos os casos e o símbolos delimitadores não fazem parte do valor. Entretanto, quando o interpretador exibe um string, ele tem que decidir qual símbolo usar para fazer com que ele se pareça um string.

```
print('Este e um string.')
print("""E este tambem e.""")
```

ActiveCode: 7 (ch02_7)

Run Save Load

Os projetista da linguagem Python usualmente decidem delimitar seus strings por apóstrofos. O que você acha que aconteceria se o string já possui-se um apóstrofo?

Quando você digita um inteiro grande, você poderia ficar tentado a usar vírgulas (N.T. em países de língua inglês) ou ponto (N.T. em países de língua portuguesa) entre grupos de três dígitos, como em `42,000` ou `42.000`. Esses não são inteiros legítimos em Python, mas têm outros significados, que são legitimos:

```
print(42000)
print(42,000)
print(42.000)
```

ActiveCode: 8 (ch02_8)

Run Save Load

Bem, isto não é de maneira alguma o que esperávamos! Devido à vírgula, Python decide tratar `42,000` como um *par* de valores. Já, no caso do ponto, Python trata `42.000` como uma representação como *float* do número 42. De fato, a função print pode imprimir qualquer número de valores, contanto que estejam separados por vírgulas. Observe que os valores são separados por espaços quando são exibidos.

```
print(42, 17, 56, 34, 11, 4.35, 32)
print(3.4, "hello", 45)
```

ActiveCode: 9 (ch02_8a)

Run Save Load

Lembre-se de não escrever os seus números inteiros com vírgulas, pontos ou espaços, não importa quão grande eles sejam. Também reveja o que dissemos no capítulo anterior: linguagens formais são rigorosas, a notação é concisa, e mesmo com a menor das modificações o resultado pode significar algo bem diferente do que você pretendia.

Teste seu entendimento

2.1.1: Como você pode determinar o tipo de uma variável?

- a) Imprimindo o valor e determinando o tipo de baseado no valor exibido.
- b) Usando a função type.
- c) Usando o valor em uma equação conhecida e imprimindo o valor resultante.
- d) Olhando para a declaração da variável.

Check Me

2.1.2: Qual é o tipo do valor 'que tipo de dado é esse'?

- a) caractere
- b) inteiro
- c) float
- d) string

Check Me

Funções para conversão de valores

Algumas vezes é necessário converter valores de um tipo para o outro. Python fornece alguns funções simples que permitirão que façamos isso. As funções `int`, `float` e `str` irão (tentar) converter seus argumentos para os tipos `int`, `float` e `str`, respectivamente. Nós as chamamos de funções para **conversão de valores**.

A função `int` pode converter para int um argumento numérico em ponto flutuante ou um string. Para números em ponto flutuante, a parte decimal do número é *descartada* – um processo que chamaremos de *trucamento para zero* ou simplesmente *truncamento* do número. Vejamos isto isto em ação:

```
print(3.14, int(3.14))
print(3.9999, int(3.9999))        # Isto não arredon
print(3.0,int(3.0))
print(-3.999,int(-3.999))         # Observe que o re

print("2345",int("2345"))         # examina um strin
print(17,int(17))                 # int também funci
print(int("23garafas"))
```

ActiveCode: 10 (ch02_20)

Run Save Load

O último exemplo mostra que o string deve representar um número sintaticamente legal, em caso contrário você receberá um daqueles erros de execução desagradáveis. Modifique o exemplo removendo `garafas` e execute novamente o programa. Você deverá ver o inteiro `23`.

O conversor de tipos `float` transforma um inteiro, um float ou um string representado um float de maneira sintaticamente legal em um float.

```
print(float("123.45"))
print(type(float("123.45")))
```

ActiveCode: 11 (ch02_21)

Run Save Load

O conversor de tipo `str` transforma os seus argumentos em um string. Lembre-se que quando imprimimos um string, os apóstrofes ou aspas que o delimitam são removidos. Entretanto, se imprimimos o tipo de um string vemos que ele é `str`.

```
print(str(17))
print(str(123.45))
print(type(str(123.45)))
```

ActiveCode: 12 (ch02_22)

Run Save Load

Teste seu entendimento

2.2.1: Qual valor é exibido pelo seguinte comando:

```
print( int(53.785) )
```

- a) Nada, é produzido um erro de execução.
- b) 53
- c) 54
- d) 53.785

Check Me

Variáveis

Uma das características mais poderosas de uma linguagem de programação é sua capacidade de manipular **variáveis**. Uma variável é um nome que se refere a um valor.

Comandos de atribuição (*assignment statement*) criam uma nova variável e também fornecem a elas o valor ao qual farão referência.

```
mensagem = "O que ha velhinho?"
n = 17
pi = 3.14159
```

Este exemplo faz três atribuições. A primeira atribui o string `"O que há velhinho?"` a uma nova variável chamada `mensagem`. O segundo atribui o inteiro `17` a `n`, e o terceiro atribui o número em ponto-flutuante `3.14159` a variável chamada `pi`.

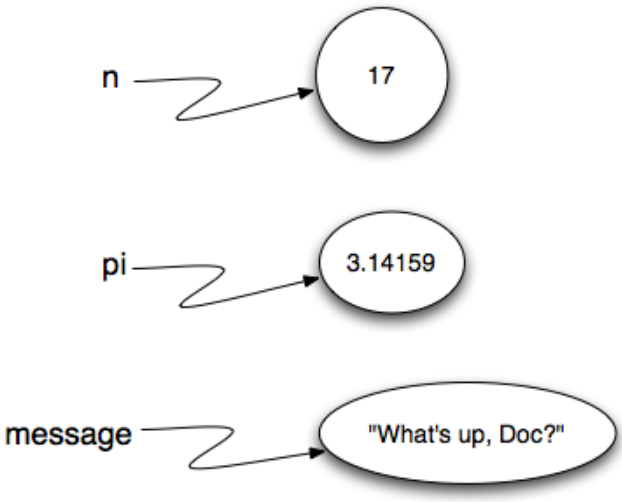
O **operador de atribuição**, `=`, não deve ser confundido com *igualdade*, para a qual usamos `==`. O comando de atribuição associa o *nome*, que está à esquerda do operador, como o *valor*, que está à direita. Por esta razão é que você receberá um mensagem de erro se fizer:

```
17 = n
```

Tip

Quando estiver lendo ou escrevendo um programa, diga para você mesmo “17 é atribuído a n” ou “n recebe o valor 17” ou “n é uma referência ao objeto 17” ou “n se refere ao objeto 17”. Não diga “n é igual a 17”.

Uma maneira comum de se representar variáveis no papel é escrevendo o nome da variável com uma flecha apontado para o valor da variável. Este tipo de representação, conhecido como **diagrama de referência**, é frequentemente chamado de **estado instantâneo** pois mostra o estado de cada variável em um instante de tempo particular. (Imagine isto como sendo o “estado de mente” da variável.) Este diagrama mostra o resultado da execução de comandos de atribuição.



Se você pedir para que o valor de uma variável seja impresso, Python exibirá o valor que está atualmente associado à variável. Em outras palavras, ao imprimir uma variável será exibido o valor ao qual a variável se refere.

```
mensagem = "O que ha velhinho?"
n = 17
pi = 3.14159

print(mensagem)
print(n)
print(pi)


```

ActiveCode: 13 (ch02_9)

Run Save Load

Em cada comando o resultado será o valor da váriável. Para ver isto em mais detalhes, podemos executar o programa usando codelens.

```
1 mensagem = "O que ha velhinho?"
2 n = 17
3 pi = 3.14159
4
5 print(mensagem)
6 print(n)
7 print(pi)
```

<< First < Back Step 1 of 6 Forward > Last >>

line that has just executed

next line to execute

Program output:

Frames Objects

CodeLens: 1 (ch02_9_codelens)

Agora, como você pode executar um comando por vez, você pode ver as variáveis e os valores a que elas se referem a medida que são criadas.

Variáveis também tem tipos; novamente, podemos perguntar ao interpretador o tipo das variáveis.

```
1 mensagem = "O que ha velhinho?"
2 n = 17
3 pi = 3.14159
4
5 print(type(mensagem))
6 print(type(n))
7 print(type(pi))
8
9
```

ActiveCode: 14 (ch02_10)

Run Save Load

O tipo de uma variável é o tipo do objeto a que ela está se referindo no momento.

Usamos variáveis em programas para “lembrar” coisas, como o placar atual de um jogo de futebol. Mas variáveis são *variáveis*. Isto significa que elas podem ser alterados ao longo do tempo, exatamente como o placar de um jogo de futebol. Você pode atribuir um valor a uma variável e mais tarde atribuir um valor diferente a mesma variável.

Note

Isso é diferente do que ocorre em matemática. Em matemática, se você dá a `x` o valor 3, esse valor não pode ser alterado durante os seus cálculos.

Para verificar isso, leia e execute o seguinte programa. Você notará que mudamos o valor da variável `dia` três vezes e na terceira vez atribuímos um valor de um tipo diferente dos anteriores

```
1 dia = "quinta-feira"
```

```
2 print(dia)
3 dia = "sexta-feira"
4 print(dia)
5 dia = 21
6 print(dia)
```

<< First | < Back | Step 1 of 6 | Forward > | Last >>

line that has just executed

next line to execute

Program output:

Frames Objects

CodeLens: 2 (ch02_11)

Uma tarefa grande em programação diz respeito a fazer o computador lembrar coisas, e.g. *O número de chamadas perdidas do seu telefone*, e atualiza ou modificar a variável quando uma nova chamada é perdida.

Teste seu entendimento

2.3.2: Qual é o valor impresso ao final da seguinte sequência de comandos?

```
dia = "quinta-feira"
dia = 32.5
dia = 19
print(dia)
```

- a) Nada é impresso, ocorre um erro de execução.
- b) quinta-feira
- c) 32.5
- d) 19

Check Me

Nomes de variáveis e palavras reservadas

Nomes de variáveis podem ser arbitrariamente longos. Eles podem conter letras e dígitos, mas eles devem começar com uma letra um caractere underscore. Apesar de ser possível usar letras maiúsculas, por convenção não usaremos. Se você usar, lembre-se que a letra ser maiúscula ou minúscula faz diferença. `Beto` e `beto` são variáveis diferentes.

O caractere underscore (_) pode aparecer no nome. Ele é usado frequentemente em nomes formados por mais de uma palavra, como `meu_nome` ou `preço_do_chá_na_china`. Existem algumas situações em que os nomes começando com um underscore têm um significado especial, portanto é mais seguro que iniciantes usem variáveis que começam com uma letra.

Se você der a uma variável um nome ilegal, ocorrerá um erro de sintaxe. No exemploa seguir, cada nome de variável é ilegal.

```
76trombones = "grande parada"
mais$ = 1000000
class = "Ciencia da Computacao 101"
```

O nome `76trombones` é ilegal pois não começa com uma letra. Já `mais$` é ilegal pois contém um caractere ilegal, o símbolo de cifrão. Mas o que está errado com `class`?

Ocorre que `class` é uma das **palavras reservadas** (*keywords*) de Python. As palavras reservadas definem a sintaxe da linguage e sua estrutura e não podem ser usadas como nomes de variáveis. Python tem pouco mais de trinta palavras reservadas (e uma vez ou outra melhorias em Python introduzem ou eliminam uma ou duas):

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

Você pode desejar mater está lista à mão. Se o interpretador reclamar sobre um dos nomes de suas variáveis e você não sabe a razão, veja se ele está nesta lista.

Caution

Iniciantes algumas vezes confundem “significado para leitores humanos” com “significativo para o computador”. Assim, eles imaginarão erroneamente que ao chamarem uma variável de `média` ou `pi`, ele irá de alguma maneira automática calcular a média ou automaticamente associará a vaŕiável `pi` com o valor 3.14159. Não! O computador não associa um significado semântico aos nomes de veriáveis.

Assim, você encontrará professores que deliberadamente não escolhem nomes significativos de variáveis quando estão lecionando para iniciantes — não por não acharem que é um bom hábito, mas porque eles estão tentando enfatizar a mensagem que você, o programador, deve escrever o código de programa para calcular a média, ou que você deve escrever um comando de atribuição para dar a uma variável o valor que você deseja que ela receba.

Teste seu entendimento

2.4.1: Verdadeiro ou falso: o seguinte nome é legal para uma variável em Python: Uma_boa_nota_é_A+

- a) Verdadeiro
- b) Falso

Check Me

Comandos e expressões

Um **comando** (*statement*) é uma instrução que o interpretador Python pode executar. Até agora só vimos o comando de atribuição. Outros tipos de comando que veremos em breve são o comando `while`, o comando `for`, o comando `if` e o comando `import`. (Existem outros tipos também!)

Uma **expressão** (*expression*) é uma combinação de valores, variáveis, operadores e chamadas de funções. Expressões necessitam ser calculadas. Se você pde ao Pyhton que

```
print(1 + 1)
print(len("Ola"))


```

ActiveCode: 15 (ch02_13)

Run Save Load

Neste exemplo `len` é uma função nativa (*built-in*) no Python que returna o número de caracteres em um string. Vimos anteriormente que as funções `print` e `type`, logo este é o nosso terceiro exemplo de uma função!

O *cálculo de uma expressão* (*evaluation of an expression*) produz um valor, que é a razão do expressão poder aparecer do lado direito de em um comando de atribuição. Um valor por si só é uma expressão e o mesmo para uma variável. Calcular o valor de uma variável resulta no valor ao qual a variável se refere.

```
y = 3.14
x = len("Ola")
print(x)
print(y)


```

ActiveCode: 16 (ch02_14)

Run Save Load

Se dermos uma olhada neste exemplo simples do Python shell, veremos uma das diferenças entre comando e expressões.

```
>>> y = 3.14
>>> x = len("Ola")
>>> print(x)
3
>>> print(y)
3.14
>>> y
```

```
3.14
>>>
```

Note que quando entramos com o comando de atribuição, `y = 3.14`, somente o prompt é retornado. Não existe valor. Isto é devido ao fato de que comando, como comando de atribuição, não retornam valor algum. Eles são simplesmente executados.

Por outro lado, o resultado da execução de um comando de atribuição é a criação de uma referência da variável, `y`, para o valor, `3.14`. Quando executamos a função print com `y` como argumento, nós vemos o valor ao qual y se refere. De fato, digitando apenas `y` obtermos o mesmo resultado.

..operator, operand, expression, integer division

Operadores e operandos

Operadores são símbolos especiais que representam computações como adição, multiplicação e divisão. Os valores sobre os quais o operador trabalha são chamados operandos.

As seguintes expressões são legais em Python e os seus significados são mais ou menos claros:

```
20 + 32
hora - 1
hora * 60 + minutos
minutos / 60
5 ** 2
(5 + 9) * (15 - 7)
```

Os símbolos `+`, `-`, `*` e o uso de parênteses têm o mesmo significado em Python do que têm em matemática. O asterisco (`*`) é i símbolo usado para indicar multiplicação, e o `**` é os ímbolo da exponenciação. Adição, subtração, multiplicação e exponenciação fazem o que você espera.

```
print(2 + 3)
print(2 - 3)
print(2 * 3)
print(2 ** 3)
print(3 ** 2)
```

ActiveCode: 17 (ch02_15)

Run Save Load

Quando o nome de uma variável aparece no lugar de um operando, ele é substituido pelo valor a que ele se refere antes da operação ser realizada. Por exemplo, veja o que pode ser feito se desejamos converter 645 minutos em horas;

```
minutos = 645
horas = minutos / 60
print(horas)
```

ActiveCode: 18 (ch02_16)

Run Save Load

Em Python 3, operador de divisão usa o símbolo `/` que sempre apresenta o resultado em ponto flutuante.

No exemplo anterior, suponha que desejamos agora saber o número de horas *cheias* e quantos minutos restantes temos em 645 minutos. Python oferece divisão de dois sabores diferentes. O segundo é chamado de **divisão inteira** (*integer division*) e usa os operador `//`. Ele sempre *trunca* o resultado para o menor inteiro (à esquerda da linha real).

```
print(7 / 4)
print(7 // 4)
minutos = 645
horas = minutos // 60
print(horas)
```

ActiveCode: 19 (ch02_17)

Run Save Load

Tome cuidado para escolher o operador de divisão correto. Se você está trabalhando com um expressão que necessita de ponto flutuante, use o operador `/`. Se você deseja um resultado inteiro use `//`.

O **operador módulo** (*modulus operator*), também chamado de **operador resto** (*remainder operator*) ou **operador resto da divisão** (*integer remainder operator*) trabalho sobre valores inteiros ou em ponto flutuante e devolve o resto da divisão inteira do primeiro operando pelo segundo. In Python, o operador resto utiliza o símbolo de porcentagem (`%`). A sintaxe é a mesma da dos outros operadores

```
quociente = 7 // 3  # divisão inteira
print(quociente)
resto = 7 % 3
print(resto)
```

ActiveCode: 20 (ch02_18)

Run Save Load

Assim, 7 dividido por 3 é 2 com resto 1.

O operador resto é surpreendentemente útil. Por exemplo, você pode utilizá-lo para verificar se um número é divisível por outro — se `x % y` é zero, então `x` é divisível por `y`. Também, você pode extrair o dígito ou dígitos mais à direita de um número. Por exemplo, `x % 10` é o dígito mais a direita de `x` (na base 10). Similarmente `x % 100` é o número formao pelos dois último dígitos de `x`.

Finalmente, retornando ao nosso exemplo de tempo, o operador resto é extremamente útil para fazermos conversões, digamos, de segundos para horas, minutos e segundos. Se começamos com um

certo número de segundos, digamos 7684, o programa a seguir usa divisão inteira e resto de divisão para converter segundos para uma forma mais clara. Siga o código passo a passo para se certificar que você entende como os operadores divisão e resto são usados para computar os valores corretos.

```
1 total_segs = 7684
2 horas = total_segs // 3600
3 segs_restantes = total_segs % 3600
4 minutos =  segs_restantes // 60
5 segs_restantes_final = segs_restantes  % 60
```

<< First | < Back | Step 1 of 5 | Forward > | Last >>

line that has just executed

next line to execute

Frames Objects

CodeLens: 3 (ch02_19_codelens)

Teste seu entendimento

2.6.1: O que imprime o seguinte comando?

```
print (18 / 4)
```

- a) 4.5
- b) 5
- c) 4
- d) 2

Check Me

2.6.2: O que imprime o seguinte comando?

```
print (18 // 4)
```

- a) 4.25
- b) 5
- c) 4
- d) 2

Check Me

2.6.3: O que imprime o seguinte comando?

```
print (18 % 4)
```

- a) 4.25
- b) 5
- c) 4
- d) 2

Check Me

Input

O program da seção anterior funciona corretamente, mas é muito limitado pois somente trabalha com o valor `total_segs`. E se desejássemos reescrever o programa de maneira que ele fique mais geral. Uma coisa que poderíamos fazer é permitir o usuário entrar com qualquer número de segundos. O programa então imprimiria o resultado apropriado para para esse valor inicial.

Para fazermos isto necessitamos de uma maneira para receber valores (*input*) do usuário. Felizmente, Python possui uma função nativa para realizar essa tarefa. Essa função é chamada `input`.

```
n = input("Por favor, digite o seu nome: ")
```

A função imput permite que apresentemos um texto ou **prompt** ao usuário (*prompt string*). Quando a função é executada o texto é exibido. O usuário da programa pode digitar o nome e pressionar a tecla *enter*. Quando isto ocorre o texto que foi digitado é retornado pela função *input* e, no presente caso, atribuído à variável *n*.

```
1 n = input("Por favor, digite o seu nome: ")
2 print("Ola", n)
3
4
```

ActiveCode: 21 (inputfun)

Run Save Load

```
Ola laiton
```

Mesmo que você pessa ao usuário para digitar a sua idade, você receberá como resposta um string como `"17"`. Será o se trabalho, como programador, converter esse string para int ou float, usando as funções de conversão *int* ou *float* que vimos anteriormente.

```
1  # leia o numero de segundos
2  segundos_str = input("Por favor, digite o numero de
3  total_segs = int(segundos_str)
4
5  # calcule o numero de horas
6  horas = total_segs // 3600
7  segs_restantes = total_segs % 3600
8
9  # calcule o numero de minutos
10 minutos =  segs_restantes // 60
11
12 # calcule o numero de segundos
13 segs_restantes_final = segs_restantes  % 60
14
15 print("Hrs=", horas, "mins=", minutos, "segs=", segs
16
17
```

ActiveCode: 22 (int_secs)

Run Save Load

```
Hrs= 0 mins= 33 segs= 20
```

A variável `segundos_str` irá se referir ao atring que foi digitado pelo usuário. Como dissemos anteriormente, mesmo que esse string seja `7684`, ele é ainda um string e não um número. Para convertê-lo para um inteiro usamos a função ``int`. O resultado será referenciado por `total_segs`. Agora, cada vez que executamos o programa, você pode entrar com um novo valor para o número de segundos a serem convertidos.

Teste seu entendimento

2.7.1: O que é impresso pelo seguinte comando?

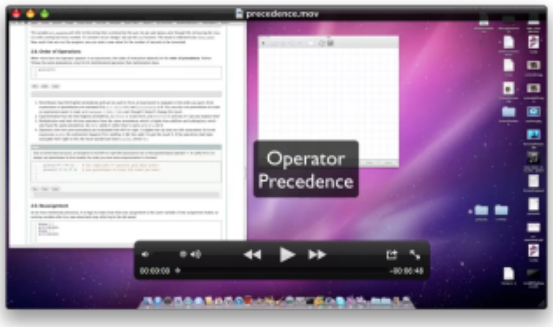
```
n = input("Por favor, digite sua idade: ")
# usuário digita 18
print ( type(n) )
```

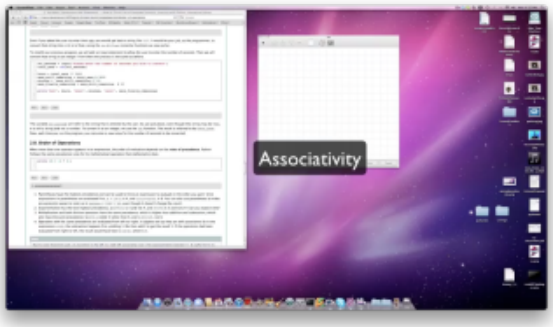
- (•) a) <class 'str'>
- ( ) b) <class 'int'>
- ( ) c) <class 18>
- ( ) d) 18

Check Me

Correct! Tudo que é digitado pelo usuário é lido como um string.

Ordem das operações





Quando mais de um operador aparece em um expressão, a ordem em que são realizadas as operações dependem das **regras de precedência** (*rules of precedence*). Python segue as regras de precedência dos seus operadores matemáticos da mesma forma que matemática.

1. Parenteses tem a mais alta precedência e podem ser usados para forçar que uma expressão seja calculada na ordem que você deseja. Como expressões entre parênteses são calculadas primeiro `2*(3-1)` é 4, e `(1+1)**(5-2)` é 8. Você

pode usar parântese para tornar uma expressão mais legível, como em `(minutos * 100) / 60`, mesmo que isto não mude o resultado.

2. Exponeciação tem a segunda precedência mais alta, assim `2**1+1``é 3 e não 4, e ``3*1**3` é 3 e não 27. Você pode explicar o por que?
3. Multiplicação e ambas as divisões têm a mesma precedência, que são mais altas que adição e subtração, que também têm a mesma precedência. Logo, `2*3-1``é 5 e não 4, e ``5-2*2` é 1 e não 6.
4. Operadores com a *mesma* precedência são executados da esquerda para a direira. Em álgebra dizemos que eles são *associativos à esquerda* (*left-associative*). Desta forma na expressão `6-3+2` a subtração é realizada primeiro e tem como resultado 3. Depois adicionamos 2 e obtemos o resultado 5. Se os operadores tivessem sido executados da direira para a esquerda o resultado seria `6-(3+2)` que é 1.

Note

Devido a alguma piculiaridade histórica, uma exceção à regra associativa à esquerda é o operador exponenciação **. Uma dica útil é sempre usar parênteses para forçar a ordem exata que você deseja quando exponenciação está envolvida.

```
print(2 ** 3 ** 2)      # o ** mais a direira e execu
print((2 ** 3) ** 2)    # use parenteses para forçar
```

ActiveCode: 23 (ch02_23)

Run Save Load

```
512
64
```

Teste seu entendimento

2.8.1: Qual é o valor da expressão a seguir?

```
16 - 2 * 5 // 3 + 1
```

- a) 14
- b) 24
- c) 3
- d) 13.667

Check Me

Correct! Usando parênteses, a éxpressão (2 * 5) é calculada primeiro, depois (10 // 3), então (16 - 3), e então (13 + 1).

2.8.2: Qual é o valor da expressão a seguir:

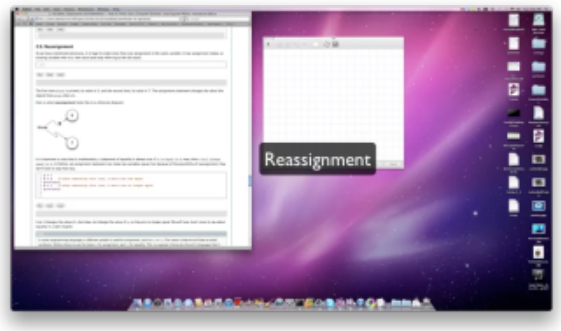
```
2 ** 2 ** 3 * 3
```

- a) 768
- b) 128
- c) 12
- d) 256

Check Me

Correct! Exponenciação tem precedência sobre multiplicação, mas a precedência é da direita para a esquerda! Assim 2 ** 3 é 8, 2 ** 8 é 256 e 256 * 3 é 768.

Reatribuição



Como mencionamos anteriormente, é válido fazer mais que uma atribuição para a mesma variável. Uma nova atribuição faz com que a variável existente se refira a um novo valor (e pare de se referir ao valor antigo).

```
bruce = 5
print(bruce)
bruce = 7
print(bruce)


```

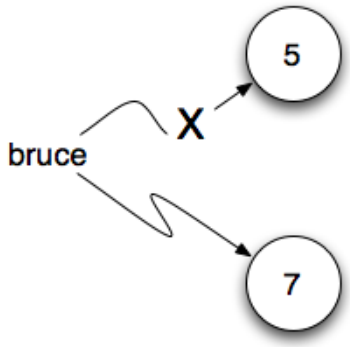
ActiveCode: 24 (ch07_reassign1)

Run Save Load

```
5
7
```

A primeira vez que `bruce` é impresso, o seu valor é 5, e na segunda vez, seu valor é 7. O comando de atribuição muda o valor (o objeto) ao qual `bruce` se refere.

Aqui está como uma **reatribuição** se parece em um diagrama de referências:



É importante notar que em matemática, uma igualdade é sempre verdadeira. Se `a é igual a b` agora, então `a será sempre igual a b`. Em Python, um comando de atribuição pode fazer duas variáveis iguais, mas devido a possibilidade de reatribuição, elas não precisam permanecer desta forma.

```
a = 5
b = a    # depois desta linha, a e b se referem a 5
print(a,b)
a = 3    # depois desta linha, b continua a se refe
```

```
5 print(a,b)
6
7
```

ActiveCode: 25 (ch07_reassign2)

Run Save Load

```
5 5
3 5
```

Linha 4 altera o valor de `a` mas não altera o valor de `b`, logo eles não são mais iguais. Teremos muito mais a dizer sobre igualdade em um capítulo mais adiante.

Note

Algumas linguagens utilizam símbolos diferentes para indicar atribuição, como `<-` ou `:=`. A intensão é evitar confusão. Python optou por usar `=` para atribuição e `==` para igualdade. Esta é uma escolha popular e também encontrada em liguagens como C, C++, Java e C#.

Teste seu entendimento

2.9.1: Depois das atribuições a seguir, quais são os valores de x e y?

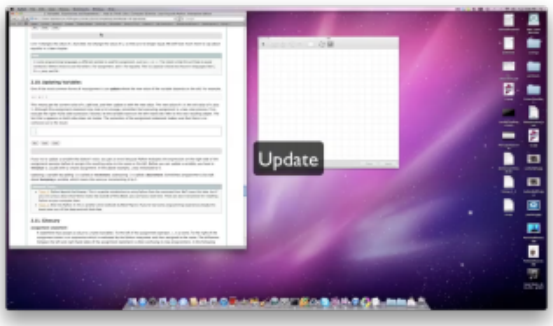
```
x = 15
y = x
x = 22
```

- a) x é 15 e y é 15
- b) x é 22 e y é 22
- c) x é 15 e y é 22
- d) x é 22 e y é 15

Check Me

Correct! Sim, x tem o valor 22 e y o valor 15.

Atualização de variáveis



Uma das forma mais comuns de reatribuição é **atualização** (*update*) onde o novo valor da variável depende do antigo. Por exemplo.

```
x = x + 1
```

Isto significa pegue o valor de x, adicione um, e atualize x com o novo valor. O novo valor de x é o anterior mais 1. Apesar desse comando de atribuição parecer um pouco estranho, lembre-se que executar uma atribuição é um processo de dois passos. Primeiro, o valor da lado direito da expressão é calculado. Segundo, faça com que o nome da variável que está no lado esquerdo se refira ao novo objeto resultante. O fato que `x` aparece em ambos os lados não

importa. A semântica do comando de atribuição se encarrega que não haja confusão sobre o resultado.

```
x = 6          # crie a variável x
print(x)
x = x + 1      # atualize x
print(x)
```

ActiveCode: 26 (ch07_update1)

Run Save Load

```
6
7
```

Se você tentar utilizar uma variável que não existe, o Python exibirá uma mensagem de erro. Considere, por exemplo, a expressão `x = x + 1`. Python calcula o valor da expressão que está do lado direito do operador de atribuição antes de atribuir o resultado ao nome sa variável do lado esquerdo. Assim, para que um erro não ocorra, é preciso que a variável `x` já tenha sido criada. Antes que você possa atualizar uma variável como a variável você deve **inicializá-la/criá-la** (*initialize* it), usualmente com uma atribuição simples. No exemplo anterior, `x` foi inicializada com 6.

Atualizar uma variável adicionando-se 1 é chamado de **incremento** (*increment*); subtrair 1 é chamado **decremento** (*decrement*). Alguns programadores também falam sobre **bumping** uma variável, que também significa incrementá-la de 1.

Advanced Topics

- Topic 1: Python além do Navegador. Está é uma introdução gentil ao uso de Python a partir da linha de comando. Veremos isto mais adiante, entretanto se você está curiosos sobre como é Python fora do contexto deste livro eletrônico, você pode dar uma olhada aqui. Também há instruções para a instalação de Python em seu computador.
- Topic 2: Dive Into Python 3, este é um livro online escrito por Mark Pilgrim. Se você teve alguma experiência prévia de programação esse livro leva você mais a fundo com os dois pés.

Teste o seu entendimento

2.10.1: O que é impresso pelo comando a seguir?

```
x = 12
x = x - 1
print (x)
```

- a) 12
- b) -1
- c) 11
- d) Nada. Ocorre um erro pois x não pode igual a x - 1

Check Me

Correct! Sim, o comando atribui a x o valor atual mesnos 1.

Scratch Editor

Open Editor

Glossário

avaliar (*evaluate*)

Simplificar uma expressão realizando as operações em ordem para obter um valor simples.

classe (*class*)

Veja **tipo de dado** (*data type*) abaixo

comando (*statement*)

Instrução que o interpretador Python pode executar. Até agora vimos apenas o comando de atribuição, mas logo encontraremos outros comandos como `import` e `for`.

comando de atribuição (*assignment statement*)

Um comando que atribui um valor a um nome (variável). À esquerdo do operador de atribuição, `=`, fica o nome. Á direita do símbolo de atribuição fica a expressão que é calculada pelo interpretador Python e é atribuído ao nome. A diferença entre os lados esquerdo e direito do comando de atribuição é sempre confuso para os novos programadores. Na atribuição a seguir:

```
n = n + 1
```

`n` tem papeis bem diferentes em cada um dos lados do `=`. Do lado direito ´´n´´'é um *valor* e faz parte da *expressão* que será calculada pelo interpretador Python antes de ser atribuído ao nome do lado direito.

comentário (*comment*)

Informação em um programa que dirigido a outros programadores (ou qualquer um que esteja lendo o código fonte) e não tem efeito algum na execução do programa.

decremento (*decrement*)

Decrescer de 1.

diagrama de referência (*reference diagram*)

Um figura mostrando uma variável com um flecha apontado para o valor (objeto) ao qual a variável se refere. Veja também ** A picture showing a variable with an arrow pointing to the value (object) that the variable refers to. See also **estado instantâneo** (*state snapshot*).

divisão inteira (*integer division*)

um operador que divide um inteiro por outro e retorna um número inteiro. Divisão inteira resulta no número de vezes que o numerador é divisivel pelo denominador e discarta qualquer resto.

estado instantâneo (*state snapshot*)

Uma representação gráfica de um conjunto de variáveis e dos valores aos quais elas se referem durante um instante particular da execução do programa.

expressão (*expression*)

Uma combinação de operadores e operandos (variáveis e valores) que tem valor simples como resultado. Expressão são avaliadas para dar o resultado.

float

Um tipo de dado do Python que armazena um número em *ponto flutuante*. Números em ponto flutuante são armazenados em duas partes: uma *base* e um *expoente*. Quando o número é impresso na forma padrão eles se parecem com números decimais. Cuidado com erros de arredondamento quando você usa `float` e lembresse que ele são apenas valores aproximados.

função para conversão de tipo (*type conversion function*)
: Uma função que pode converter um valor de um tipo para outro.

incremento/incrementar (*increment*)
: Substantivo e verbo, incrementar significa adicionar 1 a uma variável.

inicialização (de uma variável) (*initialization (of a variable)*)
: Inicializar uma variável é dar a ele um valor inicial. como em Python variáveis não existem até que elas recebam algum valor, elas são inicializadas quando são criadas. Em outras linguagens de programação este não é o caso, e variáveis podem ser criadas sem terem sido inicializadas, nesse caso elas tem um valor *default* ou *lixo*.

int
: Um tipo de dado do Python que contém números inteiros positivos e negativos.

palavra reservada (*keyword*)
: Uma palavra que é utilizada pelo compilador/interpretador na análise sintática do programa; você não pode usar palavras reservadas como `if`, `def` e `while` como nomes de variáveis.

nome de uma variável (*variable name*)
: Nome dado a uma variável. Em Python nomes de variáveis são uma sequência de letras (a..z, A..Z, e _) e dígitos (0..9) que começa com uma letra. Em uma prática de programação boa, nomes de variáveis devem ser escolhido de tal maneira que descrevam o seu uso pelo programa, fazendo que o programa seja *auto documentado* (*self documenting*).

objeto (*object*)
: Também conhecido como valor. Objetos são elementos fundamentais. Programas são projetados para manipular esses elementos (ou programadores dão ordens para que operações sejam realizadas sobre eles).

operador (*operator*)
: Um símbolo especial que representa um computação simples como adição, multiplicação ou concatenação de strings.

operador módulo (*modulus operator*)
: Chamado também de operador resto ou operado resto da divisão. Fornece o resto da divisão depois de uma divisão inteira.

operando (*operand*)
: Um dos valores manipulados por um operador.

prompt string
: Texto apresentado ao usuário indicando o tipo valor que se espera que seja digitado e de entrada ao programa.

regras de precedência (*rules of precedence*)

Conjunto de regras que governam a ordem em que expressões envolvendo vários operadores e operandos é avaliada/calculada.

símbolo de atribuição (*assignment token*)

`=` é o símbolo de atribuição usado por Python e não deve ser confundido com o operador matemático de comparação que usa o mesmo símbolo.

str

Tipo de dado do Python que armazena um atring de caracteres.

tipo de dado (*data type*)

Um conjunto de valores. O tipo de um valor determina como ele pode ser usado em uma expressão. Até agora, os tipos de dado que você viu são inteiros (`int`), números em ponto flutuante (`float`) e strings (`str`). program *self documenting*.

valor (*value*)

Um número ou string (ou outras coisas que veremos mais tarde) que podem ser armazenados em uma variável ou calculado por uma expressão.

variável (*variable*)

Nome que se refere a um valor.

Exercícios

1. Calcule de cabeça as seguintes expressões numéricas e depois use a janela do *active code* para verificar as suas respostas:

    1. `5 ** 2`
    2. `9 * 5`
    3. `15 / 12`
    4. `12 / 15`
    5. `15 // 12`
    6. `12 // 15`
    7. `5 % 2`
    8. `9 % 5`
    9. `15 % 12`
    10. `12 % 15`
    11. `6 % 6`
    12. `0 % 7`

```
print(5**2)
print(9*5)
print(15/12)
print(12/15)
print(12//15)
print(5%2)
print(12%15)
print(6%6)
print(0%7)

```

ActiveCode: 27 (ch02_ex1)

Run Save Load

```
25
45
1.25
```

```
0.8
0
1
12
0
0
```

2. Você olha para um relógio e são exatamente 2 da tarde. Você coloca um alarme para tocar daqui a 51 horas. A que horas o alarme ira tocar?

```
1 x = 51
2
3
4
5
6
7
```

run

3. Escreva um programa em Python que resolve a versão geral do problema acima. Peça ao usuário que entre com a hora atual (em horas) e que entre com o número de horas que deverá esperar antes do alarme tocar. Seu programa deve imprimir a hora que o alarme irá tocar.

Open Editor

4. Você terá umas férias maravilhosas que começam no dia 3, quarta-feira. Você retornará das sua férias depois de 137 noites (Uauu!). Escreva um programa que pede o dia do mês e o dia da semana em que você irá viajar e pede ainda o número de dias que você ficará de férias e imprime o dia da semana que você voltará.

Open Editor

5. Considere a sentença: *Só trabalho sem diversão faz de João em chato.* Armazene cada palavra em uma variável, então imprima a sentença em uma linha usando a função `print()`.

```
1  a = 'So'
2  b = 'trabalha'
3  c = 'sem'
4  d = 'diversao.'
5  print(a,b,c,d)
6
7
8
9
10
```

run

```
So trabalha sem diversao.
```

1. Acrescente parênteses à expressão `6 * 1 - 2` para mudar o esu valor de 4 para -6.

```
1 print(6*1-2)
2 print(6*(1-2))
3
4
5
6
```

```
7
```

run

```
4
-6
```

2. A fórmula para calcular o valor final de juros compostos (*compound interest*) é mostrada na Wikipedia como

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

Where,

- P = principal amount (initial investment)
- r = annual nominal interest rate (as a decimal)
- n = number of times the interest is compounded per year
- t = number of years

Escreva um programa em Python que atribui o valor 10000 para a variável `P`, atribui para `n` o valor 12 e atribui para `r` a taxa de juros de 8% (0.08). O programa deve pedir ao usuário o número `t` de anos. Calcule e imprima o valor final depois de `t` anos.

```
1  p = 1000
2  n = 12
3  r = 0.08
4  t = input('O emprestimo eh para quantos anos?
5  t = int(t)
6
7  a = p*((1+(r/n))**(n*t))
8  print(a)
9
10
11
```

run

```
1082.99950680751
```

3. Escreva um programa que calcula a área do círculo. O programa deve pedir ao usuário que digite o valor do raio. Em seguida o programa deve imprimir uma mensagem com a resposta.

```
1  # a = Pi * r * r
2  print('Calculando a area de um circulo: ')
3  pi = 3.1415
4  r = input('Qual o raio do circulo em metros? ')
5  r = int(r)
6  print('A area do circulo eh: ',pi * r *r, 'm2')
7
8
9
10
11
```

run

```
Calculando a area de um circulo:
A area do circulo eh:  12.566 m2
```

4. Escreva um programa que calcula a área de um retângulo. O programa deve pedir ao usuário que digite a altura e a largura do retângulo. Em seguida deve imprimir uma mensagem com a resposta.

```
print('Calculando a area de um retangulo')
altura = input('Digite a altura')
largura = input('Digite a largura')
print('A area do retangulo eh: ', int(altura)*
```

run

```
Calculando a area de um retangulo
A area do retangulo eh:  6 m2
```

5. Escreva um programa que calcula o consumo de gasolina de uma carro em quilômetros por litro. O programa deve pedir ao usuário que digite o número de quilômetros percorridos e o número de litros de gasolina consumidos. Em seguida o programa deve imprimir a resposta.

```
print('Calculando o consumo de combustivel')
quilometrosPercorrido = input('Digite a distan
litrosConsumido = input('Quantos litros consum
consumo = int(quilometrosPercorrido)/int(litros
print('O veiculo consumiu 1 litro de combustive
```

run

```
Calculando o consumo de combustivel
O veiculo consumiu 1 litro de combustivel para percorrer 12.5 quilometros
```

6. Escreva um programa que converta uma temperatura de graus Celsius para Fahrenheit.

```
# °F = °C × 1,8 + 32
print('Convertento de Celsius pra Fahrenheit')
c = input('qual a temperatura em graus Celsius')
print(c,' graus celsius, representa ',int(c)*1.8+3
```

run

```
Convertento de Celsius pra Fahrenheit
30  graus celsius, representa  86 Fahrenheit
```

7. Escreva um programa que converta uma temperatura de Farenheit para graus Celsius.

```
# C = (°F - 32) / 1,8
print('Convertento de Fahrenheit para Celsius')
f = input('qual a temperatura em graus Fahrenhe
print(f,' graus Fahrenheit, representa ',(int(
```

```
7
8
9
```

run

```
Convertento de Fahrenheit para Celsius
100  graus Fahrenheit, representa  37.77777777777778  Celsius
```

Como pensar como um Cientista da Computação »

previous | next | index