FUNDAMENTOS DE PROGRAMAÇÃO

SUMÁRIO

PRODUTIVIDADE E EXPRESSIVIDADE

A produtividade e expressividade das linguagens dinâmicas se encaixam perfeitamente com as metodologias ágeis, que nasceram do desenvolvimento de software de código aberto e defendem um enfoque mais pragmático no processo de criação e manutenção de software do que as metodologias mais tradicionais.

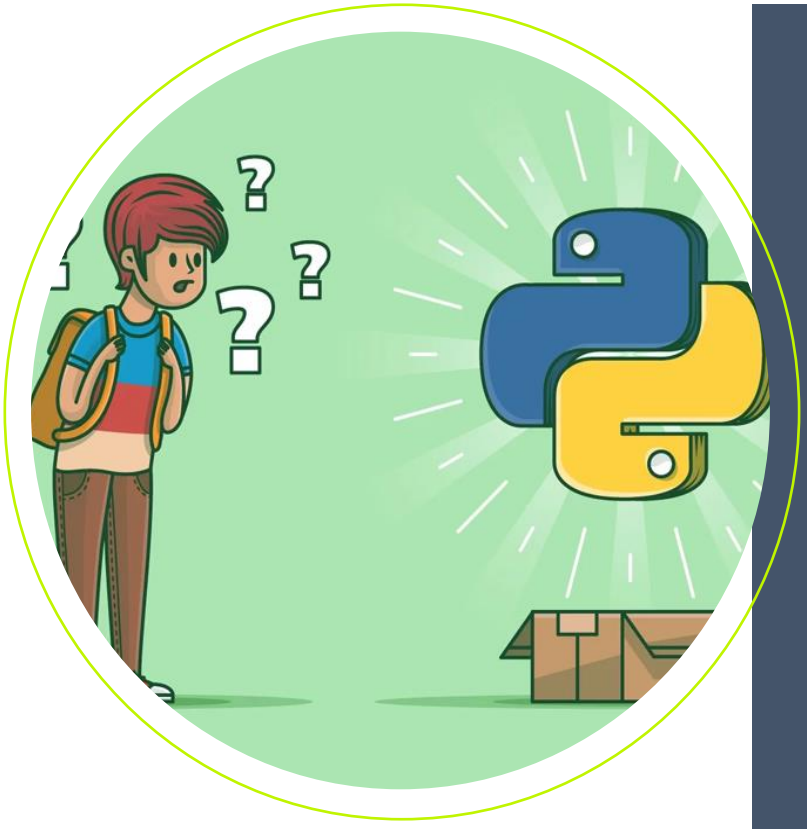
Entre as linguagens dinâmicas, o Python se destaca como uma das mais populares e poderosas. Existe uma comunidade movimentada de usuários da linguagem no mundo, o que se reflete em listas ativas de discussão e muitas ferramentas disponíveis em código aberto.

Aprender uma nova linguagem de programação significa aprender a pensar de outra forma. E aprender uma linguagem dinâmica representa uma mudança de paradigma ainda mais forte para aquelas pessoas que passaram anos desenvolvendo em linguagens estáticas.

PYTHON

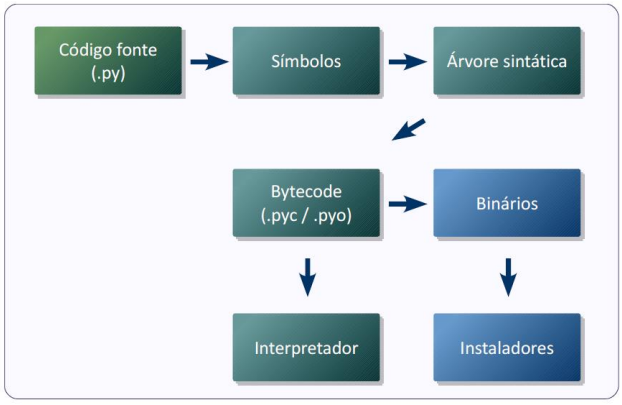
As linguagens dinâmicas eram vistas no passado apenas como linguagens script, usadas para automatizar pequenas tarefas, porém, com o passar do tempo, elas cresceram, amadureceram e conquistaram seu espaço no mercado, a ponto de chamar a atenção dos grandes fornecedores de tecnologia.

Vários fatores contribuíram para esta mudança, tais como a internet, o software de código aberto e as metodologias ágeis de desenvolvimento. A internet viabilizou o compartilhamento de informações de uma forma sem precedentes na história, que tornou possível o crescimento do software de código aberto. As linguagens dinâmicas geralmente são código aberto e compartilham as mesmas funcionalidades e em alguns casos, os mesmos objetivos.



COMPILAÇÃO E INTERPRETAÇÃO

O código fonte é traduzido pelo Python para bytecode, que é um formato binário com instruções para o interpretador. O bytecode é multiplataforma e pode ser distribuído e executado sem fonte original.



Por padrão, o interpretador compila o código e armazena o bytecode em disco, para que a próxima vez que o executar, não precise compilar novamente o programa, reduzindo o tempo de carga na execução.

Python é uma linguagem de altíssimo nível (*Very High Level Language*) orientada a objeto, de tipagem dinâmica e forte, interpretada e interativa.

ENTENDENDO

Se os arquivos fontes forem alterados, o interpretador se encarregará de regerar o bytecode automaticamente, mesmo utilizando o shell interativo. Quando um programa ou um módulo é evocado, o interpretador realiza a análise do código, converte para símbolos, compila (se não houver bytecode atualizado em disco) e executa na máquina virtual Python.

O bytecode é armazenado em arquivos com extensão ".pyc" (bytecode normal) ou ".pyo" (bytecode otimizado). O bytecode também pode ser empacotado junto com o interpretador em um executável, para facilitar a distribuição da aplicação, eliminando a necessidade de instalar Python em cada computador.

INSTALANDO

Para todos os sistemas operacionais, os instalares ficam disponíveis no link abaixo:

https://www.python.org/

Para conferir a versão do python em seu sistema operacional, digite o prompt de comando:

```
python -V
```

Atividade de Ensino.

Instale o python em seu sistema operacional e verifique a versão no prompt de comando.

BLOCOS DE CÓDIGO

Em Python, os blocos de código são delimitados pelo uso de indentação, que deve ser constante no bloco de código, porém é considerada uma boa prática manter a consistência no projeto todo e evitar a mistura de tabulações e espaços.

A recomendação oficial de estilo de codificação http://www.python.org/dev/peps/pep-0008/ é usar quatro espaços para indentação e esta convenção é amplamente aceita pelos desenvolvedores.

A linha anterior ao bloco sempre termina com dois pontos (:) e representa uma estrutura de controle da linguagem ou uma declaração de uma nova estrutura (uma função, por exemplo).

PORQUE UTILIZAR PYTHON

Uma boa pergunta é -- já que existe uma quantidade de linguagens diferentes -- por que aprender Python é importante ou mesmo interessante? Há diversas respostas; a mais importante, é que Python é fácil. É fácil em diversos sentidos.

Os conceitos fundamentais da linguagem são simples. A sintaxe da linguagem é clara e fácil de aprender. A linguagem possui um interpretador de comandos que permite aprender e testar rapidamente trechos de código. Na grande maioria dos casos, um programa em Python será muito mais curto que seu correspondente escrito em outra linguagem. Isto também faz com que seja mais rápido de escrever. Existe suporte para todo tipo de biblioteca e banco de dados possível. Ou seja, pode-se fazer em Python qualquer tipo de programa, mesmo que utilize gráficos, base SQL ou outra tecnologia externa. É possível escrever extensões a Python em C e C++ se é necessário performance máxima, ou se é desejável fazer interface com alguma ferramenta que possua biblioteca apenas nestas linguagens.

OBJETOS

Python é uma linguagem orientada a objeto, sendo assim as estruturas de dados possuem atributos (os dados em si) e métodos (rotinas associadas aos dados). Tanto os atributos quanto os métodos são acessados usando ponto (.).

Para mostrar um atributo:

```
print (objeto.atributo)
```

Para executar um método:

```
objeto.metodo(argumentos)
```

Mesmo um método sem argumentos precisa de parênteses:

```
objeto.metodo()
```

O ponto também é usado para acessar estruturas de módulos que foram importados pelo programa.

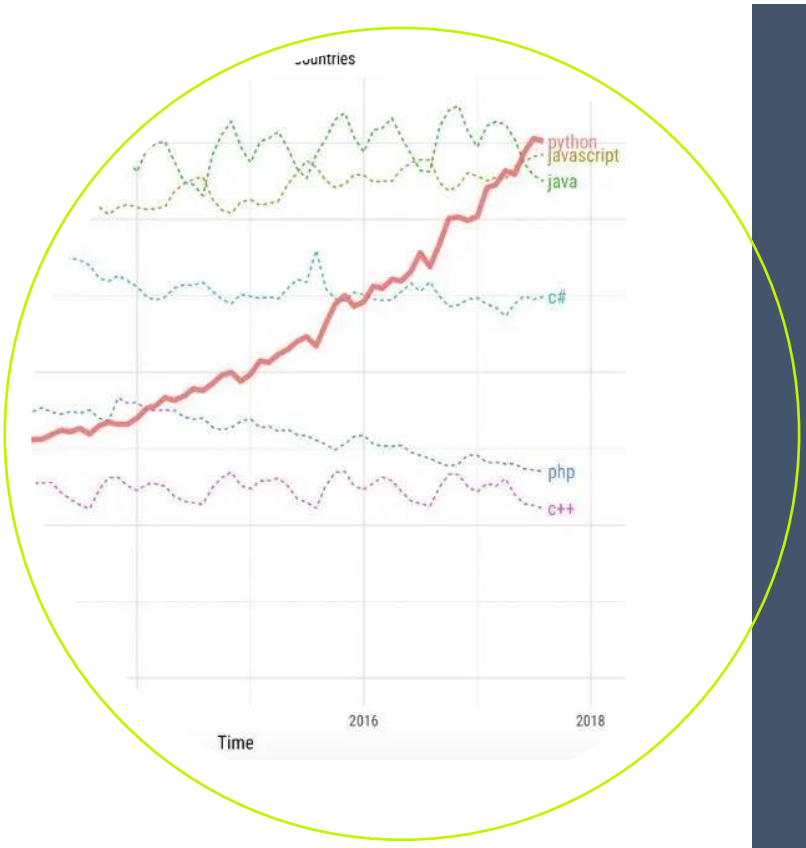
ALTO NÍVEL

Outras linguagens de alto nível de que você já pode ter ouvido falar são C++, PHP e Java. Como você pode deduzir a partir da expressão “linguagem de alto nível”, também existem as “linguagens de baixo nível”, às vezes chamadas de “linguagens de máquina” ou “linguagem assembly” (linguagens de montagem). De forma simples, o computador só consegue executar programas escritos em linguagens de baixo nível. Deste modo, programas escritos em linguagens de alto nível, precisam ser processados antes que possam rodar. Esse processamento extra toma algum tempo, o que é uma pequena desvantagem em relação às linguagens de alto nível.

Python permite que o programa funcione em múltiplas plataformas; em outras palavras, a sua aplicação feita para Linux pode rodar sem problemas em Windows e em outros sistemas.

Python é pouco punitivo: em geral, "tudo pode" e há poucas regras arbitrárias; isto acaba por tornar prazeroso o uso da linguagem.

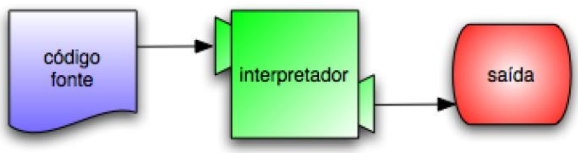
Python é um exemplo de linguagem de programação de alto nível.



PORQUE ALTO NÍVEL

Devido a essas vantagens, quase todos os programas são escritos em linguagens de alto nível. As de baixo nível são utilizadas somente para umas poucas aplicações especializadas.

Dois tipos de programas processam linguagens de alto nível, traduzindo-as em linguagens de baixo nível: interpretadores e compiladores. O interpretador lê um programa escrito em linguagem de alto nível e o executa, ou seja, faz o que o programa diz. Ele processa o programa um pouco de cada vez, alternadamente: hora lendo algumas linhas, hora executando essas linhas e realizando cálculos.



VANTAGENS

Mas as vantagens são enormes. Primeiro, é muito mais fácil programar em uma linguagem de alto nível. É mais rápido escrever programas em uma linguagem de alto nível; eles são mais curtos e mais fáceis de ler, e há maior probabilidade de estarem corretos.

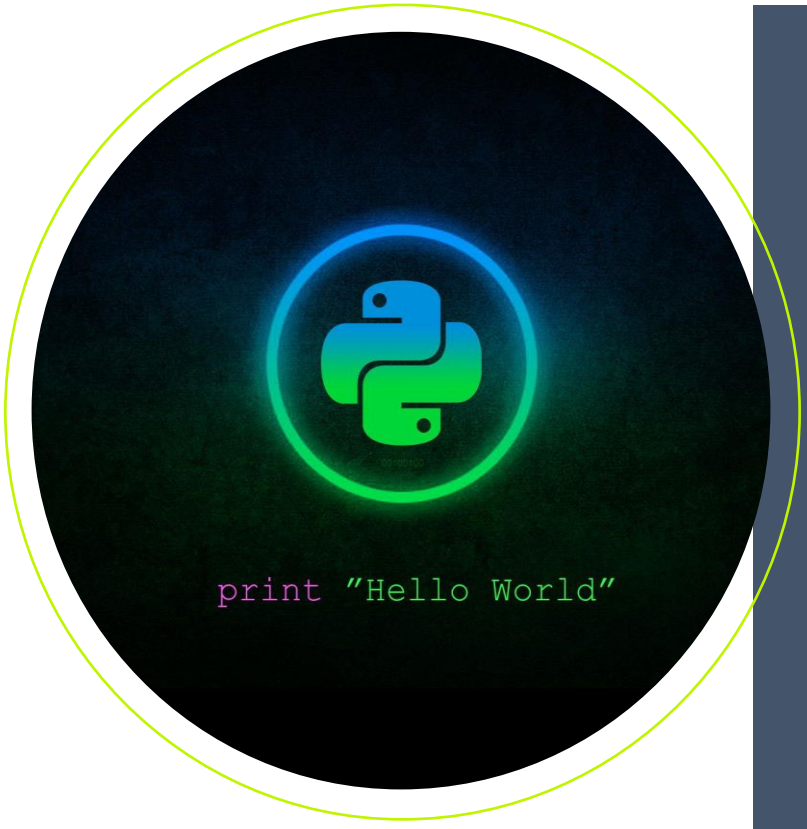
As linguagens de alto nível são portáveis, o que significa que podem rodar em diferentes tipos de computador, com pouca ou nenhuma modificação. Programas em baixo nível só podem rodar em um único tipo de computador e precisam ser reescritos para rodar em outro tipo.

Abaixo um exemplo de linguagem de baixo nível.

```
C014 24 FA          BCC    INCH       RECIEVE NOT READY
C016 B6 80 05       LDA A  ACIA+1     GET CHAR
C019 84 7F          AND A  #$7F       MASK PARITY
C01B 7E C0 79       JMP    OUTCH      ECHO & RTS

                    ****************************************
                    * FUNCTION: INHEX - INPUT HEX DIGIT
                    * INPUT: none
                    * OUTPUT: Digit in acc A
                    * CALLS: INCH
                    * DESTROYS: acc A
                    * Returns to monitor if not HEX input

C01E BD F0      INHEX  BSR    INCH       GET A CHAR
C020 81 30             CMP A  #'0        ZERO
C022 2B 11             BMI    HEXERR     NOT HEX
C024 81 39             CMP A  #'9        NINE
C026 2F 0A             BLE    HEXRTS     GOOD HEX
C028 81 41             CMP A  #'A
C02A 2B 09             BMI    HEXERR     NOT HEX
```



ENTENDENDO O CÓDIGO AO LADO

A primeira linha deste exemplo é o comando que inicia o interpretador Python. As três linhas seguintes são mensagens do interpretador. A quarta linha começa com >>>, que é o sinal usado pelo interpretador para indicar que ele está pronto. No exemplo anterior, digitamos `print (1 + 1)` e o interpretador respondeu 2. Você também pode escrever um programa em um arquivo e usar o interpretador para executar o conteúdo desse arquivo. Um arquivo como este é chamado de script. Por exemplo, usamos um editor de texto para criar um arquivo chamado marcos.py com o seguinte conteúdo:

```
print (1 + 1)
```

Por convenção, arquivos que contenham programas em Python têm nomes que terminam com .py. Para executar o programa, temos de dizer ao interpretador o nome do script:

```
python codigo.py
```

COMPILADORES

O compilador lê o programa e o traduz completamente antes que o programa comece a rodar. Neste caso, o programa escrito em linguagem de alto nível é chamado de código fonte, e o programa traduzido é chamado de código objeto ou executável. Uma vez que um programa é compilado, você pode executá-lo repetidamente, sem que precise de nova tradução.

Python é considerada uma linguagem interpretada, pois os programas em Python são executados por um interpretador. Existem duas maneiras de usar o interpretador: no modo de linha de comando e no modo de script. No modo de linha de comando, você digita programas em Python e o interpretador mostra o resultado:

```
$ python3.0

Python 3.0.1+ (r301:69556, Apr 15 2009, 15:59:22)

[GCC 4.3.3] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> print (1 + 1)

2
```

A maioria dos exemplos são executados a partir da linha de comando. Trabalhar com a linha de comando é conveniente no desenvolvimento e testagem de programas, porque você pode digitar os programas e executá-los imediatamente. Uma vez que você tem um programa que funciona, deve guardá-lo em um script, de forma a poder executá-lo ou modificá-lo no futuro.

O QUE É UM PROGRAMA?

Um programa é uma sequência de instruções que especificam como executar um cálculo ou determinada tarefa. Tal tarefa pode ser matemática, como solucionar um sistema de equações ou encontrar as raízes de um polinômio, mas também pode ser simbólica, como buscar e substituir uma palavra em um documento ou (estranhamente) compilar um programa.

Os detalhes são diferentes em diferentes linguagens, mas algumas instruções básicas aparecem em praticamente todas as linguagens:

- Entrar: Pegar dados do teclado, de um arquivo ou de algum outro dispositivo de entrada.
- Sair: Mostrar dados na tela ou enviar dados para um arquivo ou outro dispositivo de saída.
- Calcular: Executar operações matemáticas básicas, como adição e multiplicação.
- Executar condicionalmente: Checar certas condições e executar a sequência apropriada de instruções.
- Repetir: Executar alguma ação repetidamente, normalmente com alguma variação.

Acredite se quiser: isso é praticamente tudo. Todos os programas que você já usou, não importa quão complicados, são feitos de instruções mais ou menos parecidas com essas. Assim, poderíamos definir programação como o processo de dividir uma tarefa grande e complexa em subtarefas cada vez menores, até que as subtarefas sejam simples o suficiente para serem executadas com uma dessas instruções básicas.
Isso pode parecer um pouco vago, mas vamos voltar a esse tópico mais adiante, quando falarmos sobre algoritmos.

O QUE É DEPURAÇÃO

Programar é um processo complicado e, como é feito por seres humanos, frequentemente conduz a erros. Por mero capricho, erros em programas são chamados de bugs e o processo de encontrá-los e corrigi-los é chamado de depuração. (debugging). Três tipos de erro podem acontecer em um programa: **erros de sintaxe**, **erros em tempo de execução (runtime errors)** e **erros de semântica (também chamados de erros de lógica)**. Distinguir os três tipos ajuda a localizá-los mais rápido:

Erros de sintaxe
O interpretador do Python só executa um programa se ele estiver sintaticamente correto; caso contrário, o processo falha e retorna uma mensagem de erro. Sintaxe se refere à estrutura de um programa e às regras sobre esta estrutura. Por exemplo, em português, uma frase deve começar com uma letra maiúscula e terminar com um ponto.

esta frase contém um erro de sintaxe. Assim como esta

Para a maioria dos leitores, uns errinhos de sintaxe não chegam a ser um problema significativo e é por isso que conseguimos ler a poesia moderna de e. e. cummings sem cuspir mensagens de erro. Python não é tão indulgente. Se o seu programa tiver um único erro de sintaxe em algum lugar, o interpretador Python vai exibir uma mensagem de erro e vai terminar - e o programa não vai rodar. Durante as primeiras semanas da sua carreira como programador, você provavelmente perderá um bocado de tempo procurando erros de sintaxe. Conforme for ganhando experiência, entretanto, cometerá menos erros e os localizará mais rápido.

Erros em tempo de execução (runtime errors)
O segundo tipo de erro é o erro de runtime, ou erro em tempo de execução, assim chamado porque só aparece quando você roda o programa. Esses erros são também conhecidos como exceções, porque normalmente indicam que alguma coisa excepcional (e ruim) aconteceu.
Erros de runtime são raros nos programas simples, vai demorar um pouco até você se deparar com um erro desse tipo.

Erros de semântica (ou de lógica)
O terceiro tipo de erro é o erro de semântica (mais comumente chamado erro de lógica). Mesmo que o seu programa tenha um erro de semântica, ele vai rodar com sucesso, no sentido de que o computador não vai gerar nenhuma mensagem de erro. Só que o programa não vai fazer a coisa certa, vai fazer alguma outra coisa. Especificamente, aquilo que você tiver dito para ele fazer (o computador trabalha assim: seguindo ordens).

O problema é que o programa que você escreveu não é aquele que você queria escrever. O significado do programa (sua semântica ou lógica) está errado. Identificar erros semânticos pode ser complicado, porque requer que você trabalhe de trás para frente, olhando a saída do programa e tentando imaginar o que ele está fazendo.

DEBUGGING

Uma das habilidades mais importantes que você vai adquirir é a de depurar. Embora possa ser frustrante, depurar é uma das partes intelectualmente mais ricas, desafiadoras e interessantes da programação. De certa maneira, a depuração é como um trabalho de detetive. Você se depara com pistas, e tem que deduzir os processos e eventos que levaram aos resultados que aparecem.

Depurar também é como uma ciência experimental. Uma vez que você tem uma ideia do que está errado, você modifica o seu programa e tenta de novo. Se a sua hipótese estava correta, então você consegue prever o resultado da modificação e fica um passo mais perto de um programa que funciona. Se a sua hipótese estava errada, você tem que tentar uma nova. Como Sherlock Holmes mostrou: “Quando você tiver eliminado o impossível, aquilo que restou, ainda que improvável, deve ser a verdade. ” (Arthur Conan Doyle, O signo dos quatro).

Para algumas pessoas, programação e depuração são a mesma coisa. Ou seja, programar é o processo de gradualmente depurar um programa, até que ele faça o que você quer. A ideia é começar com um programa que faça alguma coisa e ir fazendo pequenas modificações, depurando-as conforme avança, de modo que você tenha sempre um programa que funciona.

Por exemplo, o Linux é um sistema operacional que contém milhares de linhas de código, mas começou como um programa simples, que Linus Torvalds usou para explorar o chip Intel 80386. De acordo com Larry Greenfield, “Um dos primeiros projetos de Linus Torvalds foi um programa que deveria alternar entre imprimir AAAA e BBBB. Isso depois evoluiu até o Linux”. (The Linux User’s Guide Versão Beta 1)

PRIMEIRO PROGRAMA

Tradicionalmente, o primeiro programa escrito em uma nova linguagem de programação é chamado de “Alô, Mundo!”, porque tudo que ele faz é apresentar as palavras “Alô, Mundo!”. Em Python, ele é assim:

```
print ("Alô, Mundo!")
```

Isso é um exemplo de um comando que faz a chamada da função print, que, na realidade, não “imprime” nada em papel. Ele apresenta o valor na tela. Neste caso, o resultado são as palavras:

Alô, Mundo!

As aspas no programa marcam o começo e o fim do valor, elas não aparecem no resultado final. Algumas pessoas julgam a qualidade de uma linguagem de programação pela simplicidade do programa “Alô, Mundo!”. Por esse padrão, Python se sai tão bem quanto possível.

VALORES E TIPOS

O valor (por exemplo, letras e números) é uma das coisas fundamentais que um programa manipula. Os valores que já vimos até agora foram o 2 (como resultado, quando adicionamos 1 + 1) e "Alô, Mundo!". Esses valores pertencem a tipos diferentes: 2 é um inteiro, e "Alô, Mundo!" é uma string, assim chamada porque “string”, em inglês, quer dizer sequência, série, cadeia (de caracteres), ou neste caso, “série de letras”. Você (e o interpretador) consegue identificar strings porque elas aparecem entre aspas.

A função print também funciona com inteiros:

```
print (4)
```

Se você estiver em dúvida sobre qual é o tipo de um determinado valor, o interpretador pode revelar:

```
type("Alô, Mundo!")
<class 'str'>
type(17)
<class 'int'>
```

Nenhuma surpresa: strings pertencem ao tipo str e inteiros pertencem ao tipo int. Menos obviamente, números com um ponto decimal pertencem a um tipo chamado float, porque estes números são representados em um formato chamado ponto flutuante:

```
type(3.2)
<class 'float'>
```

O que dizer de valores como "17" e "3.2"? Eles parecem números, mas estão entre aspas, como strings:

```
type("17")
<class 'str'>
type("3.2")
<class 'str'>
```

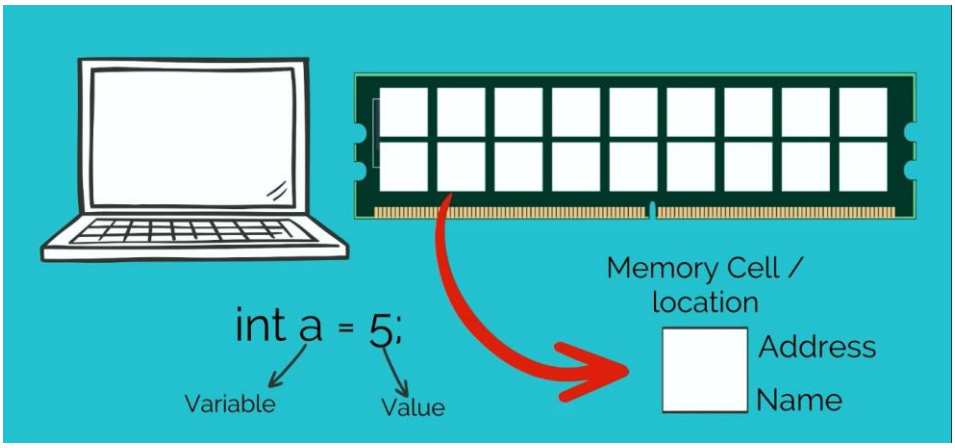
Ao digitar um número grande, é tentador usar pontos entre grupos de três dígitos, assim: 1.000.000. Isso não funciona porque Python usa o ponto como separador decimal. Usar a vírgula, como se faz em inglês, resulta numa expressão válida, mas não no número que queríamos representar:

```
print (1,000,000)
1 0 0
```

Não é nada do que se esperava! Python interpreta 1,000,000 como uma tupla, o que veremos no futuro.

VARIÁVEIS

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



O comando de atribuição cria novas variáveis e dá a elas valores:

```
mensagem = "E aí, Doutor?"
n = 17
pi = 3.14159
```

Este exemplo faz três atribuições. A primeira atribui a string "E aí, Doutor?" a uma nova variável chamada mensagem. A segunda dá o valor inteiro 17 a n, e a terceira atribui o número de ponto flutuante 3.14159 à variável chamada pi.

Uma maneira comum de representar variáveis no papel é escrever o nome delas com uma seta apontando para o valor da variável. Esse tipo de figura é chamado de diagrama de estado porque mostra em que estado cada variável está (pense nisso como o estado de espírito da variável). A função print também funciona com variáveis:

```
print (mensagem)
```
E aí, Doutor?
```
print (n)
```
17
```
print (pi)
```
3.14159

NOMES DE VARIÁVEIS E PALAVRAS RESERVADAS

Os programadores geralmente escolhem nomes significativos para suas variáveis, pois os nomes documentam para o que a variável é usada. Nomes de variáveis podem ser arbitrariamente longos. Eles podem conter tanto letras quanto números, mas têm de começar com uma letra. Embora seja válida a utilização de letras maiúsculas, por convenção, não usamos. Se você o fizer, lembre-se de que maiúsculas e minúsculas são diferentes. Bruno e bruno são variáveis diferentes.

O caractere para sublinhado (_) pode aparecer em um nome. Ele é muito utilizado em nomes com múltiplas palavras, tal como em meu_nome ou preco_do_cha_na_china.

Se você der a uma variável um nome inválido, causará um erro de sintaxe:
```
76trombones = "grande parada"
SyntaxError: invalid syntax
>>> muito$ = 1000000
SyntaxError: invalid syntax
>>> class = "Ciencias da Computacao 101"
SyntaxError: invalid syntax
```

76trombones é inválida por não começar com uma letra. muito$ é inválida por conter um caractere ilegal, o cifrão. Mas o que está errado com class?

Ocorre que class é uma das palavras reservadas em Python. Palavras reservadas definem as regras e a estrutura da linguagem e não podem ser usadas como nomes de variáveis.

Python tem 33 palavras reservadas:

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

Pode ser útil ter essa lista à mão. Se o interpretador acusar erro sobre um de seus nomes de variável e você não souber o porquê, veja se o nome está na lista. Essa lista pode ser obtida através do próprio interpretador Python, com apenas dois comandos:

```
import keyword
print (keyword.kwlist)
```

Abaixo um exemplo padrão de bloco de código, para servir de esqueleto para nossas propostas de códigos.