



Python básico

Prof. Humberto Henriques de Arruda

Descrição

Introdução à linguagem Python e suas características, apresentando as variáveis, os tipos de dados, as expressões, os operadores e as formas de atribuição, de entrada e saída de dados.

Propósito

Compreender os aspectos básicos de Python visando ao aprendizado da programação nessa linguagem.

Preparação

Será necessário baixar uma versão do Python, bem como baixar e instalar uma IDE como por exemplo, o PyCharm.

Objetivos

Módulo 1

Linguagem Python e suas características

Descrever a linguagem Python e suas principais características.

Módulo 2

Variáveis em Python

Reconhecer o uso de variáveis em Python.

Módulo 3

Tipos de dados e expressões em Python

Identificar os tipos de dados e as expressões em Python.

Módulo 4

Atribuição, entrada e saída de dados em Python

Identificar as formas de atribuição, de entrada e saída de dados em Python.

Introdução

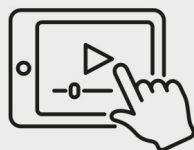
Atualmente, o ensino de disciplinas relacionadas à programação vem crescendo na educação básica e no ensino superior. Nos cursos de graduação voltados à tecnologia da informação, como Engenharia da Computação e Ciência da Computação, é frequente que se tenha contato com mais de uma disciplina focada em programação. Além desses cursos, outras carreiras têm contato com a programação em disciplinas como Introdução à Computação ou similares. Nesses diferentes contextos, várias linguagens de programação são usadas, tanto para o ensino como para o desenvolvimento de projetos.

Dentre as diversas linguagens de programação que existem, **Python** é considerada uma das principais. Por sua simplicidade de aprendizado, ela tem sido utilizada em diversos cursos universitários como a primeira linguagem com que os alunos têm contato ao programar. Atualmente, conta com ampla participação da comunidade, além de ter seu desenvolvimento aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation.

Recentemente, a [IEEE Computer Society](#) classificou-a como a linguagem mais indicada para aprender em 2020. Isso se deve à sua eficiência no desenvolvimento de *machine learning*, inteligência artificial, ciência, gestão e análise de dados.



Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Grupo voltado à ciência da computação, engenharia elétrica e eletrônica, apoiando e organizando congressos no mundo todo e divulgando artigos científicos dessas áreas.



1 - Linguagem Python e suas características

Ao final deste módulo, você será capaz de descrever a linguagem Python e suas principais características.

Visão geral



Características da linguagem Python

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Python é uma linguagem de programação de alto nível, que permite ao programador utilizar instruções de forma intuitiva, tornando seu aprendizado mais simples do que o aprendizado de uma linguagem de baixo nível.

Nas linguagens de baixo nível, o programador precisa se expressar de forma muito mais próxima do que o dispositivo “entende”, levando naturalmente a um distanciamento da linguagem utilizada para comunicação entre duas pessoas.

A linguagem Python foi lançada pelo holandês **Guido van Rossum** em 1990 e tem aumentado sua atuação na programação. Permite uma programação fácil e clara para escalas pequenas e grandes, além de enfatizar a legibilidade eficiente do código, notadamente usando espaços em branco significativos.



Características da linguagem Python

É multiparadigma

Apesar de suportar perfeitamente o paradigma de programação estruturada, também suporta programação orientada a objetos, tem características do paradigma funcional, com o amplo uso de bibliotecas, assim como permite recursividade e uso de funções anônimas.

É interativa

Permite que os usuários interajam com o interpretador Python diretamente para escrever os programas, utilizando o prompt interativo. Esse prompt fornece mensagens detalhadas para qualquer tipo de erro ou para qualquer comando específico em execução, suporta testes interativos e depuração de trechos de código.

É híbrida quanto ao método de implementação

Usa uma abordagem mista para explorar as vantagens do interpretador e do compilador. Assim como Java, utiliza o conceito de máquina virtual, permitindo a geração de um código intermediário, mais fácil de ser interpretado, mas que não é vinculado definitivamente a nenhum sistema operacional.

É portátil

Tem a capacidade de rodar em uma grande variedade de plataformas de hardware com a mesma interface. Ele roda perfeitamente em quase todos os sistemas operacionais, como Windows, Linux, UNIX, e Mac OS, sem nenhuma alteração.

É extensível

Permite que os programadores adicionem ou criem módulos e pacotes de baixo nível / alto nível ao interpretador Python. Esses módulos e pacotes de ferramentas permitem que os desenvolvedores tenham possibilidades amplas de colaboração, contribuindo para a popularidade da linguagem.

Suporta bancos de dados

Por ser uma linguagem de programação de uso geral, Python suporta os principais sistemas de bancos de dados. Permite escrever código com integração com MySQL, PostgreSQL, SQLite, ElephantSQL, MongoDB, entre outros.

Suporta interface com usuário

Permite escrever código de tal maneira que uma interface do usuário para um aplicativo possa ser facilmente criada, importando bibliotecas como Tkinter, Flexx, CEF Python, Dabo, Pyforms ou PyGUI wxPython.

Pode ser usado como linguagem de script

Permite fácil acesso a outros programas, podendo ser compilado para bytecode a fim de criar aplicativos grandes.

Permite desenvolvimento de aplicações Web

Devido à escalabilidade já citada, Python oferece uma variedade de opções para o desenvolvimento de aplicativos Web. A biblioteca padrão do Python incorpora muitos protocolos para o desenvolvimento da web, como HTML, XML, JSON, processamento de e-mail, além de fornecer base para FTP, IMAP e outros protocolos da Internet.

Permite criação de aplicações comerciais

É desenvolvido sob uma licença de código aberto aprovada pela OSI, tornando-o livremente utilizável e distribuível, mesmo para uso comercial.

Instalando o Python



Preparação do ambiente

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Instalação

Dentro do ensino aprendizagem de disciplinas relacionadas à programação, Python é uma linguagem que vem crescendo dentre as diversas linguagens de programação atuais.

Para fazer uso desta linguagem de programação bem como testes adicionais e usar as ferramentas apresentadas neste módulo será necessário baixar uma versão do Python, bem como baixar e instalar uma IDE como por exemplo, o PyCharm.

Não se preocupe ainda com o conceito de variável, nem com o seu tipo. Veremos tudo isso com detalhes nos próximos módulos deste conteúdo.

Utilitários e módulos

Apenas como exemplo, no **emulador de código** a seguir, digite `x = 5` e na linha seguinte digite `print(x, type(x))`. Depois, clique em Executar. Observe que no console ficará disponível a informação de que a variável `x` tem o valor 5 e é do tipo `int`, veja:

Exercício 1

 TUTORIAL  COPIAR

Python3

1

null

null



Esse mesmo exemplo pode ser executado no **PyCharm**, assim como os demais que serão vistos ao longo do conteúdo.

Em relação a esse exemplo, no PyCharm, na área de Console, clique no botão Python Console. No prompt interativo `>>>` que se abrirá, digite `x = 5` e pressione a tecla [ENTER] ou [RETURN] do seu teclado.

Caso você tenha alguma dúvida sobre o que é possível fazer com determinado tipo, os utilitários `dir` e `help` podem ser úteis. Veja a seguir o que cada um oferece:

Utilitário `dir`

Apresenta todos os atributos e métodos disponíveis para determinado tipo de dado.



Utilitário help

Apresenta a documentação relativa a determinado tipo de dado.

Agora é a sua vez! Teste ambos utilitários no emulador a seguir. Digite `print(dir(int))` e execute. Após isso, altere para `print(help(int))` e clique novamente em Executar, veja:

Exercício 2

TUTORIAL COPIAR

Python3

```
1
```

null



null



No prompt interativo do Python, seja no Python Console do PyCharm ou em outro ambiente Python, basta digitar `dir(int)` e `help(int)` e pressionar a tecla [ENTER] ou [RETURN], **sem a necessidade de usar a função print.**

Principais Características

Blocos

Em Python, os blocos são definidos pela indentação. Diferente de C e Java, que usam as chaves { e } para delimitar os blocos, em Python todos os blocos são iniciados com o símbolo : (dois pontos) na linha superior e representados pelo acréscimo de 4 (quatro) espaços à esquerda. Sem se preocupar por enquanto com o significado das expressões *for*, *if*, *else* ou *range*, observe os códigos abaixo:

Python

A partir disso, podemos destacar alguns pontos:

Linha 1

Está mais à esquerda, assim como as linhas 2 e 11.

Linha 2

Todas as linhas de 3 a 10 estão dentro do bloco do *for* da linha 2.

Linha 3

Observe que a linha 3 tem um *if* abrindo um bloco, dentro do qual estão as linhas 4 e 5.

Linha 6

Por sua vez, a linha 6 tem um *else* abrindo outro bloco, composto pelas linhas de 7 a 10. Os blocos do *if* (linha 3) e do *else* (linha 6) estão no mesmo nível.

Linha 7

Mostra outro *if* abrindo outro bloco – composto apenas pela linha 8 – que está no mesmo nível do bloco do *else* da linha 6 – composto apenas pela linha 10.

Linha 11

Como a linha 11 está no mesmo nível da linha 2, ela não faz parte do bloco do *for*.

Comentários

Em Python, os comentários podem ser de uma linha ou de várias linhas. A tabela a seguir mostra as formas de limitar um comentário, além de comparar essas formas em Python e C. Observe:

	Python	C
Comentários com uma linha	Iniciados com #	Iniciados com //
Comentários com várias linhas	Limitados por """ (três aspas duplas) no início e no fim	Iniciados com /* e encerrados com */

Tabela: Comentários.

Humberto Henriques de Arruda.

É importante lembrar o seguinte:

Atenção!

Os comentários não são instruções a serem executadas. Então, você pode escrever de forma simples e objetiva, sem obedecer às regras de sintaxe da linguagem.

Boas práticas de programação

Ao começar sua jornada como programador, é importante perceber que existem algumas práticas que não são obrigatórias, mas podem ajudar muito no seu aprendizado. Além disso, podem permitir que você corrija mais rapidamente erros que podem surgir no futuro e tornam seu código mais fácil de ser compreendido por outro programador, favorecendo o trabalho em equipe. Vamos conhecer algumas dessas boas práticas:



Uma prática muito importante é utilizar comentários no seu programa, explicando o que aquele trecho resolve.



Uma característica marcante da comunidade de desenvolvedores Python é manter uma lista de propostas de melhorias, chamadas [PEP](#).

EP

Sigla de Python Enhancement Proposals, dentre as PEPs, destaca-se a PEP8, que estabelece um guia de estilo de programação.

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Analise as afirmativas a seguir:

- I. Python é uma linguagem livre de alto nível, orientada a objetos e de difícil leitura, pois não permite indentação de linhas de código.
- II. Python suporta a maioria das técnicas da programação orientada a objetos.
- III. A linguagem Python e seu interpretador estão disponíveis para as mais diversas plataformas.

São corretas:

A

Somente II

B Somente III

C Somente II e III

D Somente I e II

E Somente I e III

Parabéns! A alternativa C está correta.

A linguagem Python é de fácil leitura, inclusive pela indentação. Isso torna a afirmativa I falsa. As afirmativas II e III são verdadeiras.

Questão 2

(2018/IF-MT/Informática) Sobre a linguagem Python, é incorreto afirmar que:

A Suporta os paradigmas: imperativo, orientado a objetos e funcional.

B Utiliza indentação para delimitar início e fim de blocos.

C A linguagem Python é distribuída sob licença que proíbe sua incorporação em produtos proprietários.

D Python é um software de código aberto.

E A linguagem Python suporta bancos de dados.

Parabéns! A alternativa C está correta.

A linguagem Python é desenvolvida sob uma licença de código aberto aprovada pela OSI, tornando-a livremente utilizável e distribuível, mesmo para uso comercial.



2 - Variáveis em Python

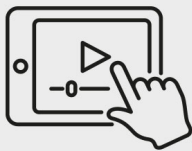
Ao final deste módulo, você será capaz de reconhecer o uso de variáveis em Python.

Variáveis



Características das variáveis em Python

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Conceitos

As variáveis são abstrações para endereços de memória que permitem que os programas fiquem mais fáceis de codificar, entender e depurar. Ao nomear uma variável com o identificador **x**, determinado espaço em memória passará a ter esse apelido. Em outras palavras, será possível acessar esse espaço de memória sabendo o seu apelido e, conseqüentemente, recuperar o valor guardado nele, que no nosso exemplo é **10**.

Uma analogia possível com o mundo real é com aquelas caixas de correio que ficam em frente às casas.



Em Python, o operador de atribuição é o = (símbolo de igual). A instrução `x = 10` atribui o valor 10 à variável `x`.

Digite `x = 10` e na linha seguinte digite `print(x)` no emulador. Em seguida, clique em Executar (no Python Console do PyCharm, digite `x = 10` e pressione a tecla [ENTER] ou [RETURN]). Depois, digite `x` e pressione [ENTER] ou [RETURN]), veja:

Exercício 1

 TUTORIAL COPIAR

Python3

1

null

null



Observe que o retorno no console foi 10.

Se, posteriormente, você digitar `x = 20` no emulador anterior e pressionar Executar (ou pressionar [ENTER] no Python Console do PyCharm), você alterará o valor da variável `x`. Ou seja, você mudará o valor armazenado naquele espaço de memória, mas sem alterar seu apelido. Faça esse teste!

Atenção!

Diferentemente de outras linguagens, como C ou Java, não é necessário declarar uma variável antes de utilizá-la em Python. Basta atribuir um valor inicial à variável e utilizá-la dali em diante. Embora não seja necessário declarar uma variável para utilizá-la, não é possível utilizar uma variável que não tenha recebido alguma atribuição de valor.

No emulador a seguir, digite `b` e pressione Executar (no Python Console do PyCharm, pressione a tecla [ENTER] ou [RETURN]), veja:

Exercício 2

 TUTORIAL COPIAR

Python3

1

null

null



Veja no emulador que a mensagem de erro informa que o nome `b` não foi definido. Ou seja, não é possível determinar o valor atribuído a esse nome.

Identificadores de variáveis

Os identificadores das variáveis podem ser compostos por letras, o **underline** (`_`) e, com exceção do primeiro caractere, números de 0 a 9. Veja os exemplos:

MinhaVariavel, _variavel, salario1 e salario1_2

São válidos.

1variavel e salario-1

Não são válidos.

MinhaVariavel e minhavariavel

São identificadores de duas variáveis distintas.

Mesmo que seja um identificador permitido, nem sempre um identificador é bom para uma variável. Tente utilizar nomes que ajudem a entender o significado da variável para ganhar tempo quando for entender o código posteriormente.

Exemplo

`salario` é um nome de variável melhor que `s`.

Algumas palavras são consideradas reservadas e não podem ser usadas como identificadores de variáveis em Python. São elas: *and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with* e *yield*.

Conceito de Amarração (*Binding*)

Amarrações

Chamamos de amarração (*binding*) a associação entre entidades de programação. Veja alguns exemplos:

- Variável amarrada a um valor;
- Operador amarrado a um símbolo;
- Identificador amarrado a um tipo.

O tempo em que a amarração ocorre é chamado de **tempo de amarração**. Cada linguagem pode ter os seguintes tempos de amarração:

Tempo de projeto da linguagem

Os símbolos são amarrados ao operador, como `*` (multiplicação), ou à definição das palavras reservadas.

Tempo de implementação



Ocorre em geral nos compiladores, como a definição de faixa de valores para determinado tipo.

Tempo de compilação



Associação da variável ao seu tipo. Lembre-se de que Python associa a variável ao tipo.

Tempo de ligação



A ligação de vários módulos compilados previamente, como a chamada a uma função de um módulo importado. Em C, utilizamos a diretiva **#include** para termos permissão de utilizar as funções de determinada biblioteca. Em Python, utilizamos o **import** para isto.

Tempo de carga



Quando o programa é carregado. Por exemplo: endereços de memória relativos são substituídos por endereços absolutos.

Tempo de execução



Associação de valores a variáveis que dependam de entradas do usuário, por exemplo. A variável é vinculada ao valor apenas durante a execução do programa.

O momento em que ocorre a ligação pode ser classificado como cedo (*early binding*) ou tardio (*late binding*). Quanto mais cedo ocorre a ligação, maior a eficiência de execução do programa, porém menor a flexibilidade das estruturas disponibilizadas.

Amarração de tipo

As amarrações de tipo vinculam a variável ao tipo do dado. Elas podem ser:

Estáticas

Ocorrem antes da execução e permanecem inalteradas. Em C, declaramos **int a**.

Dinâmicas

Ocorrem durante a execução e podem ser alteradas. É o caso do Python.

Veja a seguinte imagem:

Prompt

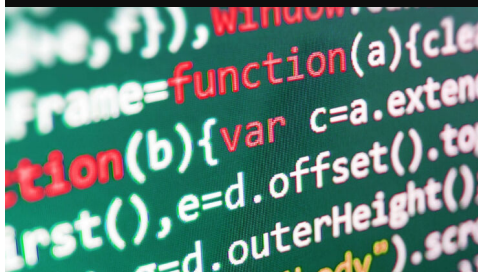


Perceba que a chamada **type** (parâmetro) retorna o tipo do parâmetro informado entre parênteses. Observe que a variável **valor** recebeu 10 e, com isso, ficou vinculada ao tipo **int**. Porém, ao receber o valor 'a', passou a estar vinculada ao tipo **str** (string).

Escopo e Tempo de Vida

Escopo de visibilidade

O escopo define em quais partes do programa uma variável é visível. Cada nome de variável em Python tem seu escopo e fora desse escopo o nome não existe, gerando um erro quando se tenta referenciar esse nome. Quanto ao escopo, chamamos as **variáveis de globais ou locais**.



Variáveis globais

Todos os nomes atribuídos no prompt interativo do Python (prompt do emulador, Python Console do PyCharm etc.) ou em um módulo fora de qualquer função são considerados como de escopo global. Por exemplo, ao executar a instrução da caixa de código a seguir, a variável **x** é uma variável global, veja:

Prompt



Variáveis locais

Para exemplificar o uso de variáveis com escopo local, vamos utilizar uma função definida pelo desenvolvedor. Não se preocupe com esse tipo de função por enquanto, você aprenderá mais detalhes posteriormente. Por enquanto, observe o emulador a seguir:

Exercício

TUTORIAL

COPIAR

Python3

```
1 def multiplicador(numero):  
2     a = 2 # esta variável tem escopo local
```

```
3     print(f"Dentro da função, a variável vale: {a}")
4     return a * numero
5
6
```

null

null



As linhas 2, 3 e 4 compõem o bloco interno à função chamada **multiplicador()**. Embora as variáveis das linhas 2 e 6 tenham o mesmo nome, elas são abstrações a endereços de memória diferentes. Dentro da função **multiplicador()**, a chamada ao nome **a** recupera o valor 2. Fora da função **multiplicador()**, a chamada ao nome **a** recupera o valor 3. Execute o código no emulador acima e veja a saída que será produzida.

Agora, observe a função **multiplicador()** com uma pequena alteração, em que retiramos a inicialização da variável **a** dentro da função.

Exercício

TUTORIAL

COPIAR

Python3

```
1 def multiplicador(numero):
2     return a * numero
3
4 a = 3 # esta variável tem escopo global
5 b = multiplicador(5)
6
```

null

null



Na linha 5, ao se chamar a função do **multiplicador()**, a variável **a** será procurada. Como não existe uma variável **a** no bloco interno da função, ela é procurada como variável global. Uma vez encontrada, o valor recuperado é 3. Ao executar esse código a saída obtida será *A variável b vale 15*. Confira por si mesmo, clique em Executar no emulador anterior.

Usamos este exemplo para mostrar que o interpretador Python pode procurar o mesmo nome de variável em diferentes escopos. A ordem utilizada para a procura é:

1. A chamada da função delimitadora;
2. Variáveis globais;
3. O módulo builtins.

Perceba que, se **a** variável **a** é inicializada na função **multiplicador()**, qualquer chamada a esse nome dentro da função resultará na referência a essa variável local. Mas seria possível alterar a variável **a** global com uma instrução dentro da função **multiplicador()**? Sim, utilizando-se a palavra

reservada **global**. Agora é a sua vez! Veja no emulador a seguir como isso poderia ser feito. Clique em Executar e obtenha o resultado do código:

Exercício

 TUTORIAL COPIAR

Python3

```
1 def multiplicador(numero):
2     global a # todas as referências à variável a são para a global
3     a = 2     # a global será alterado
4     print(f"Dentro da função, variável vale: {a}")
5     return a * numero
6
```

null

null



Escopos

Os tipos de escopo são:

Estático

O escopo é baseado na descrição textual do programa e as amarrações são feitas em tempo de compilação. É o caso de C, C++ e Java, por exemplo.

Dinâmico

O escopo é baseado na sequência de chamada dos módulos (ou funções). Por isso, as amarrações são feitas em tempo de execução. É o caso do Python.

O fato de Python ser de escopo dinâmico traz alguns problemas, como a perda de eficiência – uma vez que os tipos precisam ser verificados em tempo de execução – e a redução na legibilidade – porque é difícil determinar a sequência exata de todas as chamadas de função.

Tempo de vida

Embora escopo e tempo de vida tenham uma relação próxima, eles são conceitos diferentes. Observe:

Escopo

É um conceito textual.



Tempo de vida

É um conceito temporal.

As variáveis globais têm o tempo de vida que é o de execução do programa, ao passo que as variáveis locais somente existem no intervalo de duração da função ou do bloco a que se limitam.

Constantes

Definição

Em Python, não existe o conceito de constante. Se você precisar de uma constante ao longo de sua jornada como programador, atribua o valor a uma variável e tome cuidado para não mudar esse valor.

Dica

Inicie o nome dessa variável com `c_` ou utilize todas as letras maiúsculas, o que vai diferenciar essa variável das outras. Por exemplo, é possível utilizar a expressão `c_PI = 3.141592` para armazenar o valor de PI e agilizar o cálculo de área e perímetro de um círculo, ou utilizar a expressão `PRECISION = 0.001` para armazenar a precisão a ser utilizada em qualquer cálculo matemático no seu programa.

É importante ficar atento ao uso correto das variáveis, especialmente observando as questões de escopo e visibilidade, para evitar que algum cálculo seja realizado corretamente, mas com resultado diferente do esperado por você ao programar.

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

(IF-CE/2017/Técnico de Laboratório Informática) Considere o trecho do programa Python abaixo:

Python



Os valores impressos, ao se executar o programa, são, respectivamente:

A 1 e 1

B 10

C 1 e 10

D 10 e 10

E 10 e 1

Parabéns! A alternativa C está correta.

A variável `x` da linha 2 é local da função `func()`, sendo visível para a chamada `print()` da linha 3. Por sua vez, a variável `x` da linha 5 é global, sendo visível para a chamada `print()` da linha 7.

Questão 2

(MS CONCURSOS/2016/Creci 1° Região (RJ)/Analista de TI) Qual alternativa representa a declaração de uma variável na linguagem de programação Python?

A `var valor = 3`

B `Boolean inicio = falso`

C `Texto = 'texto de exemplo'`

D `Int i = 1`

E `not = falso`

Parabéns! A alternativa C está correta.

Lembre-se de que, em Python, as variáveis não são declaradas com o tipo vinculado. Assim, basta atribuir um valor inicial à variável para que ela possa ser usada. Isso ocorre com a variável `texto`, que recebe o valor inicial "texto de exemplo".



3 - Tipos de dados e expressões em Python

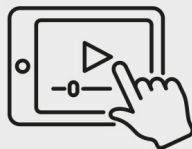
Ao final deste módulo, você será capaz de identificar os tipos de dados e as expressões em Python.

Tipos de dados padrão



Tipos sequenciais e dicionários

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Conceitos

Agora você será apresentado aos tipos de dados padrão incorporados ao interpretador Python. Os principais tipos internos são:



Numéricos



Sequenciais



Dicionários

Classes, instâncias e exceções também são tipos padrão, mas não entraremos em detalhes aqui. Para ter nosso primeiro contato com expressões em Python, use o prompt interativo `>>>`.

Ao digitar uma expressão algébrica no Python Console do PyCharm (ou em qualquer outro prompt interativo do Python) e pressionar a tecla [ENTER], o resultado da expressão é obtido. Isso ocorre porque o Python, através de seu prompt interativo, permite que você calcule expressões algébricas como uma calculadora, além de executar instruções básicas em Python. Veja a seguir:

Prompt



Tipos numéricos

Existem três tipos numéricos distintos em Python. Confira:



Números inteiros



Números de ponto flutuante



Números complexos

Lembrando que os booleanos são um subtipo dos números inteiros.

O tipo int

É o tipo usado para manipular números inteiros. Fazendo uma analogia com a Matemática, o tipo `int` é usado para elementos do conjunto dos inteiros (\mathbb{Z}).

Diferentemente de outras linguagens, como C ou Java, a linguagem Python não limita o tamanho de uma variável de qualquer tipo, logo, não existe um valor inteiro máximo definido. O limite depende da quantidade de memória disponível no computador. Novamente, fazendo uso do prompt interativo do Python, veja algumas coisas interessantes.

Digite `1_000_000` e pressione a tecla [ENTER], como no box a seguir:

Prompt



Python permite que você utilize o **underline** (_) como separador de milhar. Isso ajuda a visualizar números com muitos dígitos. Para encerrarmos este primeiro contato com o tipo int, no emulador seguinte, digite `print(type(1_000_000))`, clique em Executar e verifique que o valor do exemplo anterior é um inteiro. Esse mesmo teste pode ser feito no PyCharm, basta digitar em seu Python Console o seguinte: `type(1_000_000)` e apertar a tecla [ENTER], veja:

Exercício

 TUTORIAL  COPIAR

Python3

1

null

null



0 tipo float

É o tipo usado para manipular números com parte inteira e parte decimal, chamados de números de ponto flutuante. Fazendo uma analogia com a Matemática, o tipo **float** é usado para elementos do conjunto dos reais (**R**).

Para diferenciar um número real de um inteiro, é possível utilizar a parte decimal zerada. No emulador a seguir, digite `print(type(50.0))` e pressione Executar (No Python Console do PyCharm, digite `type(50.0)` e pressione [ENTER]), veja:

Exercício

 TUTORIAL  COPIAR

Python3

1

null

null



Vale ressaltar o seguinte:

Atenção!

Devemos usar o ponto para separar a parte inteira da parte decimal, e não a vírgula.

No emulador anterior, digite x = 50.0. Em seguida, digite y = 50,0, depois print(x) e por fim, print(y). Após isso, clique em Executar.

Ao usar a vírgula como separador em Python, o que ocorre, na verdade, é a criação de uma [tupla](#) de dois elementos, e não o tipo float.

Embora os tipos `int` e `float` sejam semelhantes, por tratarem de números, eles têm propriedades um pouco diferentes. Em expressões algébricas, sempre que somamos, subtraímos ou multiplicamos apenas elementos do tipo `int`, o resultado é `int`. Porém, basta um operando do tipo `float` para que o resultado seja `float`. Agora é a sua vez! Clique em Executar no emulador abaixo e observe o resultado no console:

tupla

Você verá mais detalhes sobre tuplas em um momento posterior.

Exercício

 TUTORIAL  COPIAR

Python3

```
1 print(type(2+3+1))
2 print(type(2+3+1.0))
```

null

null



Vamos analisar a exponenciação. Para realizar essa operação matemática, utilizamos o operador (`**`). Clique em Executar no emulador abaixo:

Exercício

 TUTORIAL  COPIAR

Python3

```
1 print (2**3)
2 print (type(2**3))
3 print (type(2.0**3))
```

null

null



Veja que basta que a base seja **float** para que o resultado também o seja.

Atenção!

Diferentemente de outras linguagens, como C, a divisão de dois números inteiros não necessariamente tem resultado inteiro.

No emulador seguinte, digite as linhas de código abaixo e depois clique em Executar:

```
x = 5/2
```

```
print(x)
```

Exercício

TUTORIAL

COPIAR

Python3

```
1
```

null

null



No PyCharm, em seu Python Console, basta digitar 5/2 e pressionar [ENTER].

Para obter o quociente inteiro e resto, quando dois inteiros são divididos, é necessário utilizar os operadores // e %, respectivamente. Ao dividir 21 por 2, temos quociente 10 e resto 1. Observe no box a seguir.

Prompt



0 tipo complex

É o tipo utilizado para manipular números complexos, na forma **x + yj**, sendo **x** a parte real e **y** a parte imaginária do complexo.

Veja dois exemplos de variáveis do tipo complex nas imagens seguintes, em que a parte real é 2 e a parte imaginária é 5:


```
Python3  
>>> r = complex(2,5)  
>>> r  
(2+5j)
```

Complex 1

```
>>> w=2+5j  
>>> type(w)  
<class 'complex'>
```

Complex 2

A chamada `r.conjugate()` retorna o conjugado do número complexo `r`, em que a parte real é mantida e a parte imaginária tem o seu sinal trocado.

O tipo bool

Uma expressão algébrica, como vimos nos exemplos dos tipos `int` e `float`, é avaliada como um número, seja desses tipos ou de outro tipo numérico admitido em Python. Porém, utilizar expressões não algébricas também é bastante comum. E uma boa notícia é que Python pode avaliar expressões desse tipo também. Essa é uma diferença entre Python e outras linguagens, como C, por exemplo, em que não existe o tipo `bool`.

No prompt interativo (PyConsole do PyCharm, por exemplo), digite a expressão `2 < 3` e pressione [ENTER]. Ou, no emulador a seguir, digite `print(2 < 3)` e clique em Executar. Observe o resultado que será obtido:

Exercício

[TUTORIAL](#) [COPIAR](#)

Python3

1

null

null



Repare que o resultado dessa expressão não é um número, mas sim a palavra `True`. Caso você colocasse a expressão `2 > 3`, o resultado seria `False`. Faça o teste no PyCharm ou no emulador anterior.

Saiba mais

As expressões que você viu nos dois exemplos são chamadas de **expressões booleanas**. Trata-se de expressões que podem ser avaliadas com um dos dois valores booleanos: **True** ou **False**. Assim, em Python, existe o tipo **bool**, utilizado para permitir o tratamento de expressões como essas.

Agora, vamos ver o operador **not**, que é um operador unário, ou seja, só precisa de um operando. Esse operador inverte o valor booleano, ou seja, se o valor original for **True**, **not(valor)** terá o valor **False**. E vice-versa.

No prompt interativo, digite a expressão `not(2 < 3)` e pressione [ENTER]. Ou, no emulador anterior, digite `print(not(2 < 3))` e clique em Executar. Verifique que o resultado a ser obtido é o contrário do obtido anteriormente.

É possível também escrever expressões booleanas compostas, utilizando conectivos como **E OU**. Vamos ver mais detalhes sobre essas expressões ainda neste módulo. Fique atento!

Operadores numéricos

Operadores matemáticos

Os operadores matemáticos são muito semelhantes àqueles que vimos ao longo de nossa jornada como estudantes, aprendendo Álgebra e Aritmética na escola. Existem algumas pequenas diferenças, como a divisão (que pode ser a usual ou a divisão inteira). Mas é possível identificar operações que fizemos ao longo de toda nossa vida. A tabela a seguir lista os operadores de expressão aritmética disponíveis em Python, veja:

Operação matemática	Símbolo usado	Exemplo	
		Equação	Resultado
Soma	+	2.5 + 1.3	3.8
Subtração	-	2.5 - 1.3	1.2
Multiplicação	*	2.5 * 1.3	3.25
Divisão	/	2.5/1.3	1.923076923076923
Divisão inteira	//	9/2	4
Resto na divisão inteira	%	9%2	1
Valor absoluto	abs(parâmetro)	abs(-2.5)	2.5
Exponenciação	**	2**4	16

Tabela: Operadores matemáticos.
Humberto Henriques de Arruda.

Além das operações algébricas, é possível realizar operações de comparação. Os operadores de comparação têm como resultado um valor booleano (**True** ou **False**). Observe a tabela abaixo:

Símbolo usado	Descrição
<	Menor que
<=	Menor ou igual a
>	Maior que

Símbolo usado	Descrição
>=	Maior ou igual a
==	Igual
!=	Não igual

Tabela: Operadores de comparação.
Humberto Henriques de Arruda.

Cabe observar o seguinte:

Atenção!

O operador utilizado para comparar se dois valores são iguais é o ==, ou seja, duplo sinal de igual. Tome cuidado para não confundir com o operador de atribuição, que é representado pelo sinal de igual apenas uma vez (=).

Existe outra lista de operadores que executam operações matemáticas, mas, além disso, atualizam o valor da variável utilizada. Eles são chamados de operadores compostos e serão detalhados no módulo 4. Para mais funções matemáticas, você pode utilizar os módulos matemáticos math e fractions.

Operadores booleanos

As expressões booleanas são aquelas que podem ter como resultado um dos valores booleanos: True ou False. É comum utilizarmos os operadores de comparação em expressões booleanas, mas não só eles.

Assim como é possível escrever expressões algébricas complexas concatenando diversas expressões menores, podemos escrever expressões booleanas grandes, com os operadores **and**, **or** e **not**. Observe o comportamento dos operadores booleanos nas três tabelas abaixo:

p	not (p)
True	False
False	True

Humberto Henriques de Arruda.

Operador not

p	q	p and q
True	True	True
True	False	False
False	True	False
False	False	False

Humberto Henriques de Arruda.

Operador and

p	q	p or q
True	True	True
True	False	True
False	True	True
False	False	False

Humberto Henriques de Arruda.

Operador or

Tipos sequenciais

Existem três tipos sequenciais básicos em Python:



Listas



Tuplas



Objetos range

Além desses, existe um tipo especial criado para tratamento de dados textuais: o tipo **str** (string).

Assim como em C ou Java, a indexação dos itens é iniciada com 0 e cada item tem o seu índice incrementado uma unidade em relação ao item anterior. Porém, Python também permite a indexação com valores negativos. O valor -1 é o índice do último item, e cada item anterior é decrementado de uma unidade em relação ao sucessor, veja:

índice	0	1	2	3	4
s	t	e	s	t	e
índice negativo	-5	-4	-3	-2	-1

Tabela: Índices em tipos sequenciais.
Humberto Henriques de Arruda.

Strings

Em uma variável do tipo str, é possível armazenar letras, números, espaços, pontuação e diversos símbolos. Diferentemente da linguagem C, não existe o tipo char. Cada caractere em Python é uma string. Para delimitar uma string, podemos utilizar:

Aspas simples

```
'uma string'
```

Aspas duplas

```
"uma string"
```

Aspas simples triplas

```
'''uma string'''
```

Aspas duplas triplas

```
"""uma string"""
```

O box a seguir ilustra um exemplo de delimitadores de strings, veja:

Prompt



Existem alguns métodos interessantes para tratar strings em Python. Entre eles, ressaltamos:

Upper

Transforma todas as letras em maiúsculas.

Lower

Transforma todas as letras em minúsculas.

Split

Quebra a string em substrings.

Veja o exemplo:

Prompt



A lista gerada com o método **split()** tem três elementos, porque a string original tinha três palavras.

Listas

Listas são sequências mutáveis, normalmente usadas para armazenar coleções de itens homogêneos. Uma lista pode ser criada de algumas maneiras, tais como:

[]

Usando um par de colchetes para denotar uma lista vazia.

[a], [a, b, c]

Usando colchetes, separando os itens por vírgulas.

[x for x in iterable]

Usando a compreensão de lista.

list() ou list(iterable)

Usando o construtor do tipo list.

Iterable pode ser uma sequência, um container que [suporte iteração ou um objeto iterador](#). Por exemplo, `list('abc')` retorna `['a', 'b', 'c']` e `list((1, 2, 3))` retorna `[1, 2, 3]`. Se nenhum argumento for passado, o construtor cria uma lista vazia: `[]`.

suporte iteração ou um objeto iterador

Um iterador é um objeto que contém um número contável de valores. Ele pode ser iterado, o que significa que podemos percorrer todos os valores.

Tuplas

Tuplas são sequências imutáveis, tipicamente usadas para armazenar coleções de itens heterogêneos. Elas são aplicadas também quando é necessário utilizar uma sequência imutável de dados homogêneos. Uma tupla pode ser criada de algumas maneiras, tais como:

()

Usando um par de parênteses para denotar uma tupla vazia.

a, b, c ou **(a, b, c)**

Separando os itens por vírgulas.

tuple() ou **tuple(iterable)**

Usando o construtor do tipo **tuple**.

Novamente, iterable pode ser uma sequência, um container que suporte iteração ou um objeto iterador. Por exemplo, **tuple('abc')** retorna ('a', 'b', 'c') e **tuple([1, 2, 3])** retorna (1, 2, 3). Se nenhum argumento for passado, o construtor cria uma tupla vazia: **()**.

Atenção!

Note que o uso das vírgulas é o que gera a tupla, e não o uso de parênteses. Os parênteses são opcionais, exceto no caso em que queremos gerar uma tupla vazia.

Range

O tipo **range** representa uma sequência imutável de números e frequentemente é usado em loops de um número específico de vezes, como o **for**.

Ele pode ser chamado de maneira simples, apenas com um argumento. Nesse caso, a sequência começará em 0 e será incrementada de uma unidade até o limite do parâmetro passado (exclusive). Por exemplo, **range(3)** cria a sequência (0, 1, 2).

Para que a sequência não comece em 0, podemos informar o início e o fim como parâmetros, lembrando que o parâmetro fim não entra na lista (exclusive o fim). O padrão é incrementar cada termo em uma unidade. Ou seja, a chamada **range(2, 7)** cria a sequência (2, 3, 4, 5, 6).

Saiba mais

Também é possível criar sequências mais complexas, indicando os parâmetros de início, fim e passo, nessa ordem. O passo é o valor que será incrementado de um termo para o próximo. Por exemplo, **range(2, 9, 3)** cria a sequência (2, 5, 8).

Operadores sequenciais comuns

Os operadores sequenciais permitem a manipulação dos tipos sequenciais, inclusive as strings. Vale ressaltar a sobrecarga dos operadores **+** e *****, que realizam operações diferentes quando os operandos são numéricos ou sequenciais.

Exemplo

O operador **==** verifica se as strings dos dois lados são iguais. Porém, os operadores **<** e **>** comparam as strings usando a ordem do dicionário.

A tabela a seguir traz um pequeno conjunto dos operadores disponíveis em Python para manipulação de sequências. Lembre-se de que você pode utilizar o utilitário `help` no Python Console para verificar a lista completa. Para isso, basta digitar `help(str)` e pressionar [ENTER] no teclado.

Uso	Resultado
<code>x in s</code>	True se x for um subconjunto de s
<code>x not in s</code>	False se x for um subconjunto de s
<code>s + t</code>	Concatenação de s e t
<code>n*s</code>	Concatenação de n cópias de s
<code>s[i]</code>	Caractere de índice i em s
<code>len(s)</code>	Comprimento de s
<code>min(s)</code>	Menor item de s
<code>max(s)</code>	Maior item de s

Tabela: Operadores sequenciais.
Humberto Henriques de Arruda.

Dicionários

Os dicionários permitem que itens de uma sequência recebam índices definidos pelo usuário. Um dicionário contém pares de (chave, valor). O formato geral de um objeto dicionário é:

Prompt

Poderíamos criar um dicionário em que cada pessoa fosse representada pelo seu CPF, com nome e sobrenome. Para isso, teríamos:

```
>>> pessoas = {'111222333-44':['João','Silva'], '222333444-55':['Maria','Oliveira'], '333444555-66':['Pedro','Costa']}
>>> pessoas['111222333-44']
['João', 'Silva']
```

Dicionários.

Na imagem anterior, o dicionário tem 3 entradas. Observe como foi possível recuperar nome e sobrenome de uma entrada, baseado na chave informada '111222333-44'.

Ainda sobre os dicionários, vamos a mais um vídeo!

Relação de precedência entre operadores

Ao escrever uma expressão algébrica, o programador pode utilizar a precedência de operadores existente em Python (implícita) ou explicitar a ordem em que ele deseja que a expressão seja avaliada.

Exemplo

A expressão `3 + 2 * 5` tem como resultado 25 ou 13? Aprendemos no ensino fundamental que as operações de produto e divisão têm precedência sobre as operações de soma e subtração. Ou seja, um produto será realizado antes de uma soma, na mesma expressão. Assim, a expressão acima tem como resultado 13. Isso ocorre sempre que não forem explicitadas outras relações de precedência com o uso de parênteses. Caso o programador quisesse forçar que a soma ocorresse primeiro, ele deveria escrever assim: `(3 + 2) * 5`.

Sempre que o programador quiser forçar a ocorrência de uma operação antes de outras, ele pode utilizar os parênteses para aumentar a prioridade sobre ela. A tabela a seguir traz as relações de precedência entre os operadores, com as linhas mais altas tendo prioridade sobre as linhas mais baixas. Ou seja, elas ocorrem primeiro. Dentro da mesma linha, a precedência é da esquerda para a direita. Observe:

Operador	Descrição
[expressões ...]	Definição de lista
x[], x[índice : índice]	Operador de indexação
**	Exponenciação
+x, -x	Sinal de positivo e negativo
*, /, //, %	Produto, divisão, divisão inteira, resto
+, -	Soma, subtração
in, not in, <, <=, >, >=, <=>, !=, ==	Comparações, inclusive a ocorrência em listas
not x	Booleano NOT (não)
and	Booleano AND (e)
or	Booleano OR (ou)

Tabela: Operadores sequenciais.
Humberto Henriques de Arruda.

É importante ficar atento ao uso correto dos operadores, respeitando a precedência entre eles, para evitar que algum cálculo seja realizado corretamente, mas com resultado diferente do esperado por você ao programar.

Conversões de tipos de dados

Quando temos tipos diferentes envolvidos na mesma expressão, o Python converte implicitamente cada operando para o tipo mais abrangente envolvido na expressão. Estamos usando a palavra abrangente, mas poderíamos falar que existem tipos que englobam (ou contêm) outros.

Exemplo

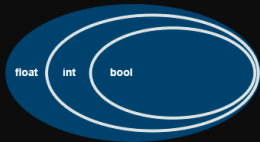
Um número do tipo **int** pode ser visto como um **float** com a parte decimal nula. Porém, o inverso não é verdade. Ou seja, o conjunto dos inteiros (**int**) é um subconjunto do conjunto dos reais (**float**). Assim, a expressão **5 + 0.68** – que envolve um **int** e um **float** – tem como resultado **5.68**. O inteiro 5 é convertido pelo Python para o número de ponto flutuante 5.0 antes que a soma (de dois valores **float**) seja realmente efetuada.

Uma conversão implícita não intuitiva é a dos valores booleanos **True** e **False** em inteiros, respectivamente, 1 e 0. Veja o exemplo:

Prompt



Com isso, podemos perceber a seguinte relação entre os tipos **bool**, **int** e **float**:



Relação entre os tipos bool, int e float.

Além das conversões implícitas, o programador também pode usar as conversões explícitas, quando ele força que o valor seja tratado como de determinado tipo. Para isso, é necessário usar o construtor do tipo desejado, com o valor passado como parâmetro (entre parênteses). Veja o exemplo:

Prompt



O **int** 2 pode ser tratado naturalmente como o **float** 2.0, basta acrescentar a parte decimal nula. Porém, ao tentar tratar um **float** como **int**, ocorre a remoção da parte decimal.

Atenção!

Fique atento, porque não é uma aproximação para o inteiro mais próximo, e sim o truncamento.

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Considere a expressão a seguir: $2 + 3 - 4 ** 2 + 5 / 2 - 5 // 2$

Assinale a opção com o valor correto dessa expressão em Python.

A -10.5

B -1

C 1.5

D 2

E 1

Parabéns! A alternativa A está correta.

Lembre-se de que o operador `**` tem precedência maior do que os operadores `/` e `//`, os quais, por sua vez, têm precedência sobre `+` e `-`. Ou seja, primeiro será efetuada a exponenciação ($4**2$), depois as divisões, comum ($5/2$) e inteira ($5//2$), para posteriormente serem efetuadas as somas e subtrações.

Questão 2

(Adaptada de COMPERVE/2019/UFRN/Engenharia da Computação) Python é uma linguagem interpretada muito utilizada. Não requer tipagem de variáveis e sua sintaxe indentada favorece a organização do código. Uma das suas funcionalidades mais poderosas são as listas. Considere o código em Python do quadro abaixo:

Python



A saída correta correspondente às linhas 2 e 4 do código é

A 2 e 4.

B

4 e 16.

C

2 e 16.

D

4 e 4.

E

2 e 2.

Parabéns! A alternativa A está correta.

O operador + realiza operações de soma para tipos numéricos e concatenação para tipos sequenciais. Assim, a variável **a** na linha 1 passa a ser composta dos itens 'UF' e 'RN'. Assim, a chamada **len(a)** retorna o tamanho 2, número de elementos de **a**. De forma semelhante, o operador * realiza operações de multiplicação para tipos numéricos e concatenação de cópias para tipos sequenciais. Assim, a variável **b** na linha 3 passa a ser a lista ['4', '4', '4', '4']. E a chamada **len(b)** retorna o tamanho 4, número de elementos de **b**.



4 - Atribuição, entrada e saída de dados em Python

Ao final deste módulo, você será capaz de identificar as formas de atribuição, de entrada e saída de dados em Python.

Variáveis: formas de atribuição



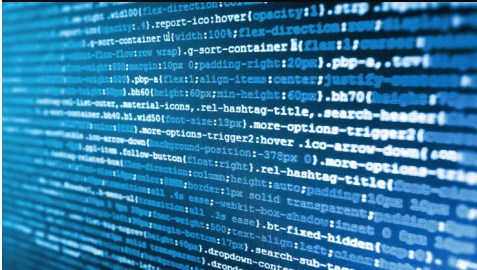
Seu primeiro programa e exercícios

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Conceitos

Já vimos no módulo 2, de maneira básica, como podemos atribuir valor a uma variável. Vamos agora conhecer outras formas de atribuição.



Sentenças de atribuição

Atribuição simples

Chamamos de atribuição simples a forma que já utilizamos neste conteúdo, com uma expressão parecida com `x = 10`. Nessa atribuição, a variável `x` recebe o valor 10.

Atribuição múltipla

Python também permite a atribuição múltipla, ou seja, mais de uma variável receber atribuição na mesma linha. A título de exemplo, clique em Executar no emulador abaixo:

Exercício

TUTORIAL

COPIAR

Python3

```
1 x, y = 2, 5
2 print(x)
3 print(y)
```

null



null



Observe que as variáveis `x` e `y` receberam atribuição na mesma instrução, com a variável `x` armazenando o valor 2, e a variável `y` armazenando o valor 5.

Vamos avançar!

Operadores de atribuição compostos

Os operadores de atribuição compostos executam operações matemáticas e atualizam o valor da variável utilizada. Veja no código abaixo (clique em Executar):

Exercício

TUTORIALCOPIAR

Python3

```
1 x = 10
2 x = x + 1
3 print(x)
```

null

null



A variável `x`, inicialmente, recebeu o valor 10. Em seguida, a instrução `x = x + 1`, que causa estranheza quando lembramos da matemática aprendida ao longo da vida, é muito comum quando estamos programando. Essa instrução significa “acrescente uma unidade ao valor de `x` e guarde este resultado na própria variável `x`”. Como `x` valia 10, o resultado do lado direito do operador (`=`) é 11. Esse resultado é, então, armazenado na própria variável `x`.

Essa operação de acrescentar determinado valor a uma variável e armazenar o resultado na própria variável poderia ser feita com o operador `+=` (mais igual). Altere a 2ª linha no código do emulador anterior para `x += 1`, clique em Executar e veja o resultado.

Na tabela a seguir, estão os operadores compostos disponíveis em Python. Considere a variável `x`, com o valor inicial 10, para verificar os resultados:

Nome	Símbolo usado	Exemplo	
		Instrução	Resultado
Mais igual	<code>+=</code>	<code>x += 2</code>	<code>x</code> passa a valer 12
Menos igual	<code>-=</code>	<code>x -= 2</code>	<code>x</code> passa a valer 8
Vezez igual	<code>*=</code>	<code>x *= 2</code>	<code>x</code> passa a valer 20
Dividido igual	<code>/=</code>	<code>x /= 2</code>	<code>x</code> passa a valer 5
Módulo igual	<code>%=</code>	<code>x %= 3</code>	<code>x</code> passa a valer 1

Tabela: Operadores compostos.
Humberto Henriques de Arruda.

Diferente de C, em Python não é possível incrementar ou decrementar uma variável com um operador unário, como o `++` ou `--`.

Troca de variáveis

Um dos problemas iniciais que envolvem atribuição de valores a variáveis é a troca entre duas delas. Suponha que as variáveis `a` e `b` armazenem, respectivamente, os valores 1 e 2. Caso quiséssemos inverter os valores em linguagens como C ou Java, seria necessário usar uma variável auxiliar. Em Python, é possível fazer essa troca de uma maneira muito mais fácil, com o uso da atribuição múltipla. Veja no código abaixo (clique em Executar) as duas maneiras de se trocar valores entre 2 variáveis, através do uso de variável auxiliar e através de atribuição múltipla, veja:

Exercício

TUTORIAL

COPIAR

Python3

```
1 a = 1
2 b = 2
3 # troca de variáveis usando variável auxiliar 'temp'
4 temp = a
5 a = b
6 b = temp
```

null

null

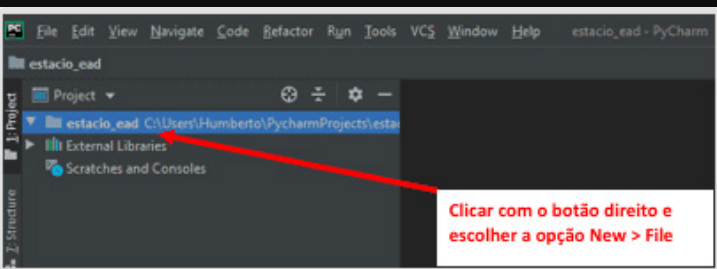


Programação em Python

O primeiro programa em Python

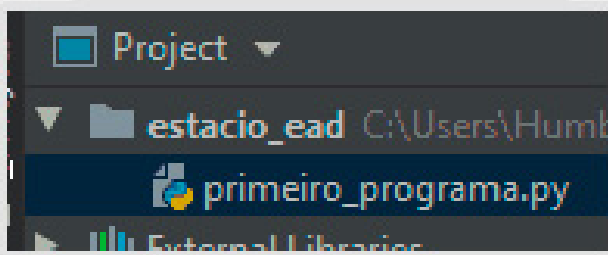
Para escrever um programa em Python, será essencial utilizar as formas de **saída de dados** para exibir ao usuário mensagens e resultados de operações. Caso você deseje que o usuário informe algum dado para que seu programa processe, será necessário utilizar as formas de **entrada de dados**.

Para criar seu primeiro programa, utilize o PyCharm. Nele, clique com o botão direito do mouse no nome do projeto, na guia de navegação do lado esquerdo. Em seguida, escolha a opção `New > File`, como na seguinte imagem:



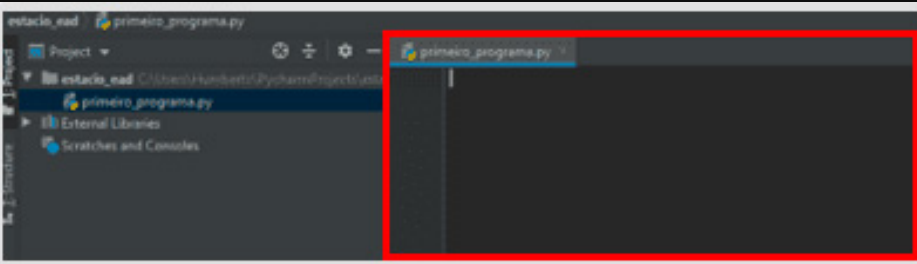
Novo arquivo no PyCharm.

Atribua um nome a seu arquivo. Neste primeiro exemplo, vamos chamar de `primeiro_programa.py`, veja:



Primeiro_programa.py.

Ao nomear o arquivo, será aberta uma nova aba do lado direito, com o espaço para que você efetivamente digite as instruções, veja:



Aba de codificação.

Saída de dados com a função print()

A função **print()** em Python atua de forma semelhante à **printf()** em C. Para um programador iniciante, as maiores diferenças entre elas são:

Duas chamadas da print() em Python

São impressas na tela em linhas diferentes, sem a necessidade do uso do caractere '\n' para pular a linha, como ocorre na printf() em C.



Uma chamada da print() em Python

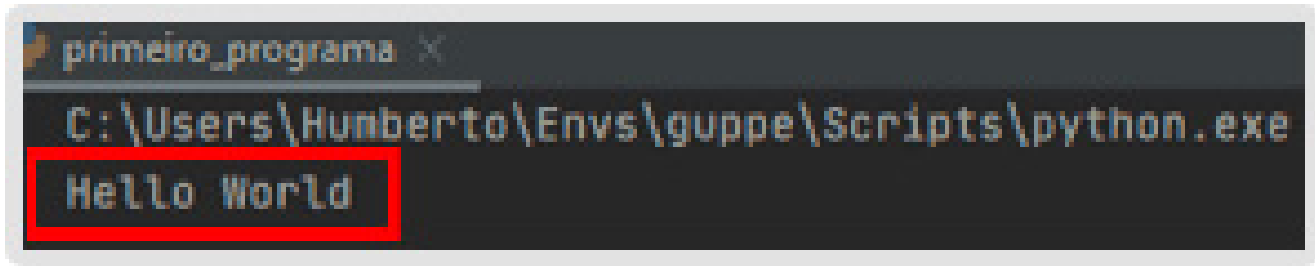
Permite a impressão de valores de variáveis sem a indicação do formato, como ocorre na printf() em C, quando precisamos escrever %c, %d ou %f, por exemplo.

Para escrever seu Hello World em Python, digite a seguinte linha, exatamente como está escrita:

Python



Em seguida, clique com o botão direito do mouse sobre o nome do programa e escolha a opção Run 'primeiro_programa'. Também é possível executar com a combinação de teclas CTRL+Shift+ F10. Após executar, observe o console na imagem abaixo:

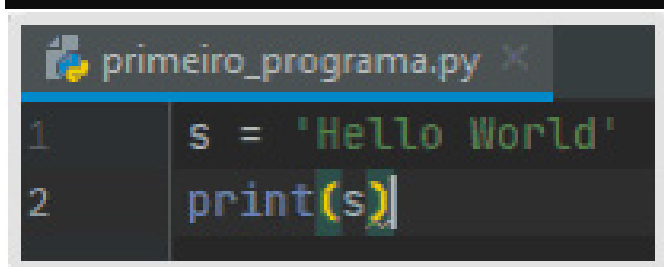


```
primeiro_programa x
C:\Users\Humberto\Envs\guppe\Scripts\python.exe
Hello World
```

Execução do Hello World em Python.

Veja que foi impresso no console exatamente o que colocamos entre aspas, ao chamar a função **print()**. Essa é a primeira forma de saída de dados: usar a função **print()** com uma string sendo passada como parâmetro (entre os parênteses). É importante perceber que a função **print()**, além de imprimir a string, também salta o cursor para a próxima linha.

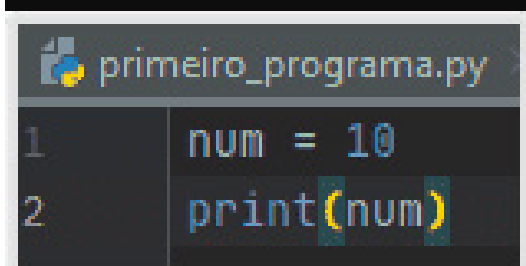
Como você deve ter percebido, o que a função **print()** recebeu entre parênteses foi uma string. Ou seja, poderíamos ter passado para ela uma string já definida. Veja no exemplo:



```
primeiro_programa.py x
1 s = 'Hello World'
2 print(s)
```

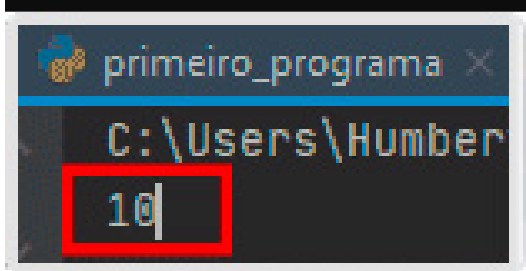
Hello World com string.

Também poderíamos ter passado como parâmetro uma variável já definida. A função **print()** vai trabalhar com o valor dessa variável. Observe as imagens seguintes:



```
primeiro_programa.py x
1 num = 10
2 print(num)
```

Print de variável.

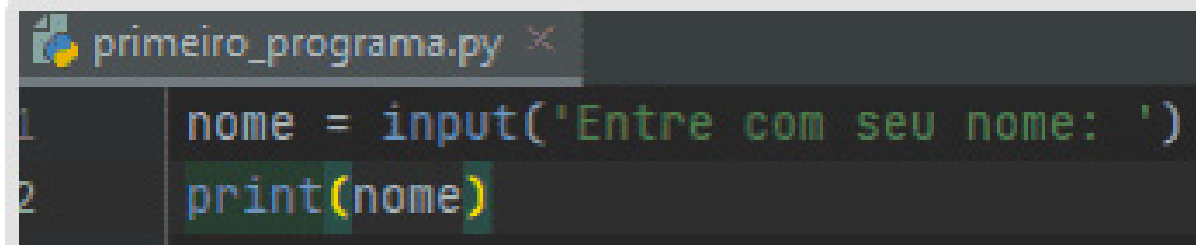


```
primeiro_programa x
C:\Users\Humber
10
```

Execução do print com variável.

Entrada de dados com a função Input()

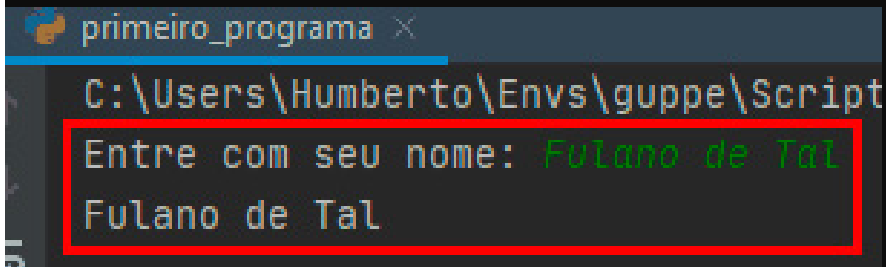
Quando o programador quiser que o usuário entre com algum valor, ele deverá exibir na tela o seu pedido. Em C, é necessário utilizar a função **printf()** para escrever a solicitação ao usuário e a função **scanf()** para receber a entrada e armazenar em uma variável. Em Python, é possível utilizar a função **input()**. Ela tanto exibe na tela o pedido, como permite que o valor informado pelo usuário seja armazenado em uma variável do seu programa. Analise a imagem seguinte:



```
primeiro_programa.py x
1 nome = input('Entre com seu nome: ')
2 print(nome)
```

A função input().

A linha 1 fará com que a frase **Entre com seu nome:** seja exibida no console, mas a execução do programa fica travada até que o usuário aperte [ENTER] no teclado. Tudo o que foi digitado até o [ENTER] vai ser armazenado na variável **nome**. A linha 2 fará a exibição do conteúdo da variável **nome**. Veja o resultado no console, com o usuário tendo digitado **Fulano de Tal**:



```
primeiro_programa x
C:\Users\Humberto\Envs\guppe\Script
Entre com seu nome: Fulano de Tal
Fulano de Tal
```

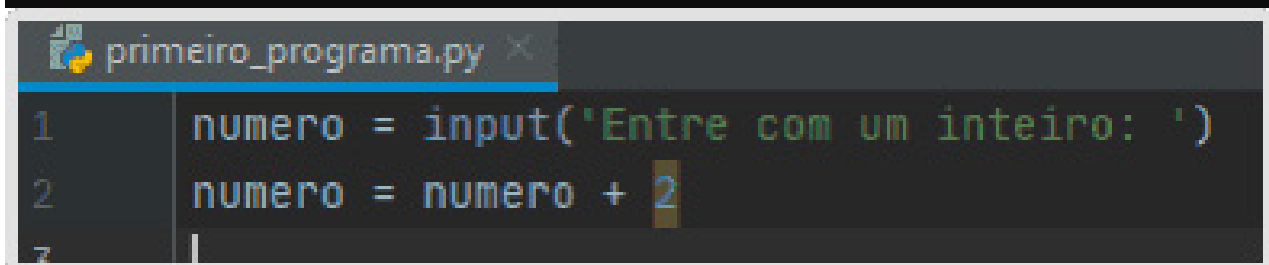
Execução da entrada de dados.

Vale lembrar que:

Atenção!

É importantíssimo perceber que a função input() trata tudo o que for digitado pelo usuário como uma string, armazenando na variável designada pelo programador para isso. Mesmo que o usuário entre com apenas uma letra ou um número, isso será armazenado como uma string na variável.

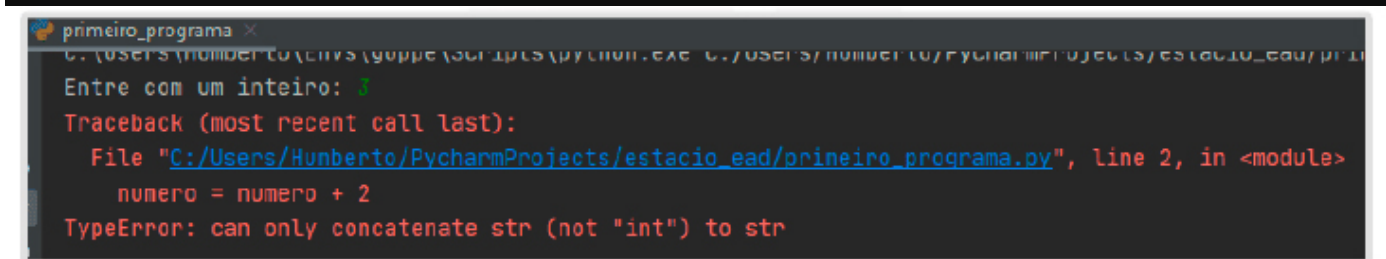
Vamos então a mais um exemplo:



```
primeiro_programa.py x
1 numero = input('Entre com um inteiro: ')
2 numero = numero + 2
```

A função input() e operação numérica.

Veja o console quando o programa é executado:



```
primeiro_programa x
C:\Users\Humberto\Envs\guppe\Scripts\python.exe C:/Users/Humberto/PycharmProjects/estacio_ead/primeiro_programa.py
Entre com um inteiro: 3
Traceback (most recent call last):
  File "C:/Users/Humberto/PycharmProjects/estacio_ead/primeiro_programa.py", line 2, in <module>
    numero = numero + 2
TypeError: can only concatenate str (not "int") to str
```

Erro com a função input().

O usuário digitou 3 e [ENTER]. Mesmo sendo um valor, a variável **numero** trata como a string '3'. Isso impede que seja realizada a operação de soma com o inteiro 2, por exemplo. Poderíamos também usar a instrução **print(type(numero))** na linha 2 para confirmar, veja:

```
primeiro_programa.py <
1 numero = input('Entre com um inteiro: ')
2 print(type(numero))
```

Função print() com função type().

```
primeiro_programa <
C:\Users\Humberto\Envs\g
Entre com um inteiro: 3
<class 'str'>
```

Resultado de print() com type().

A função eval()

A função **eval()** recebe uma string, mas trata como um valor numérico. Veja o exemplo:

```
Python Console
>>> s = '1 + 2'
>>> type(s)
<class 'str'>
>>> eval(s)
3
```

Função eval().

Mesmo tendo recebido a string '1+2' como parâmetro, a função **eval()** efetuou a soma de 1 com 2. Observe que confirmamos que s é uma string com a instrução **type(s)**.

Para tratar a entrada do usuário como um número e, com isso, realizar operações algébricas, por exemplo, é necessário utilizar a função **eval()** em conjunto com a **input()**, veja:

```
primeiro_programa.py <
numero = eval(input('Entre com um inteiro: '))
numero = numero + 2
print(numero)
```

Função eval() com função input().

```
primeiro_programa <
C:\Users\Humberto\Envs\g
Entre com um inteiro: 3
5
```

Resultado do uso de `eval()` e `input()`.



Atividade discursiva

Como exercício prático, tente escrever um programa para calcular e informar o IMC (índice de massa corpórea) do usuário, que deverá fornecer seu peso e sua altura. Lembre-se de que o IMC é calculado pela fórmula: $IMC = \frac{\text{peso}}{\text{altura}^2}$

Digite sua resposta aqui

Exibir solução ▾

Uma solução simples é demonstrada na seguinte imagem:



```
1 peso = eval(input('Entre com o seu peso: '))
2 altura = eval(input('Entre com a sua altura: '))
3 imc = peso/(altura**2)
4 print('IMC = ', imc)
```

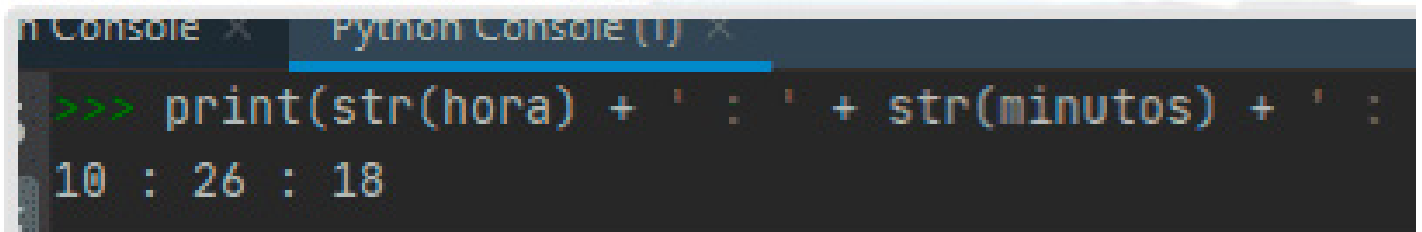
Código do exercício.

Formatação de dados de saída

Quando desejamos que a saída siga determinado padrão – por exemplo, de hora ou de data – existem algumas possibilidades para usar a função `print()`. É sempre possível utilizar a concatenação de strings, com o operador `+`, para montar a frase como quisermos. Suponha que tenhamos as seguintes variáveis:

```
hora = 10
minutos = 26
segundos = 18
```

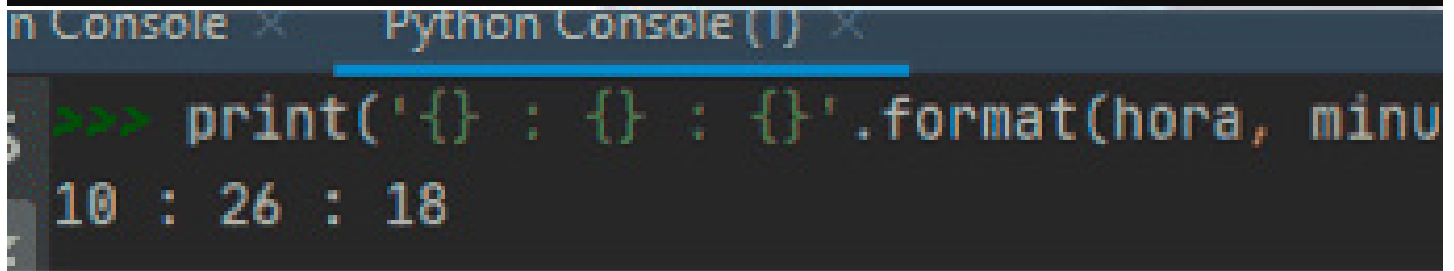
Poderíamos chamar a função `print()` com o separador da seguinte forma:



```
>>> print(str(hora) + ' : ' + str(minutos) + ' : ' + str(segundos))
10 : 26 : 18
```

Saída de dados com separador.

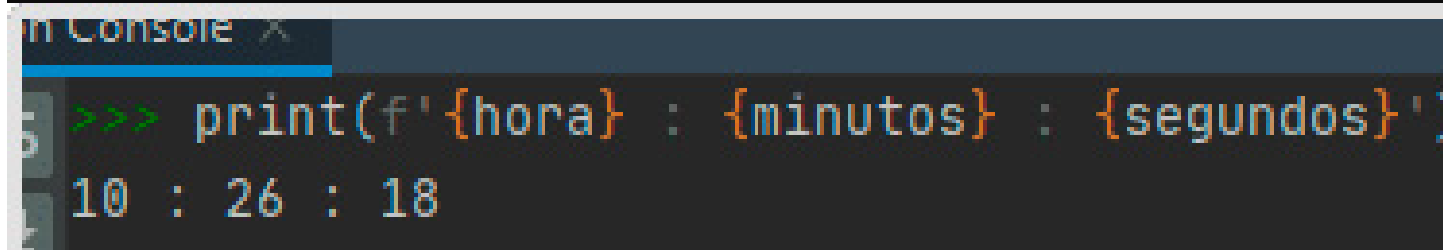
Porém, existe outra possibilidade, usando o método **format()**. Ele permite que a chamada à função **print()** fique muito parecida com as chamadas à função **printf()** em C, com passagem de parâmetros a serem colocados em ordem na string. Com o método **format()**, podemos montar a string com as chaves **{}** indicando onde entrarão valores, passados como parâmetros separados por vírgulas, como mostrado na imagem abaixo:



```
>>> print('{} : {} : {}'.format(hora, minutos, segundos))
10 : 26 : 18
```

Saída de dados com **format()**.

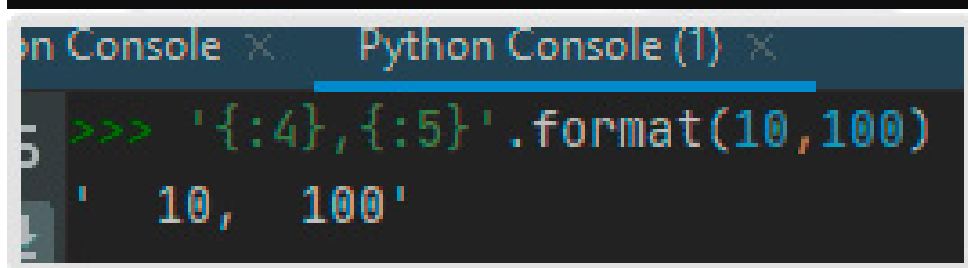
É importante observar que a quantidade de chaves precisa ser igual à quantidade de variáveis passadas como parâmetros no método **format()**. Seria muito bom se não precisássemos nos preocupar com essa correspondência para evitar erros bobos. E isso é possível! Para tornar a saída formatada ainda mais intuitiva, basta utilizar a letra **'f'** no início dos parâmetros da função **print()** e colocar cada variável dentro das chaves na posição em que deve ser impressa. Veja como fica ainda mais fácil entender:



```
>>> print(f'{hora} : {minutos} : {segundos}')
10 : 26 : 18
```

Saída de dados com **print()** e **f**.

Também é possível especificar a largura de campo para exibir um inteiro. Se a largura não for especificada, ela será determinada pela quantidade de dígitos do valor a ser impresso. Confira na imagem:



```
>>> '{:4},{:5}'.format(10,100)
' 10, 100'
```

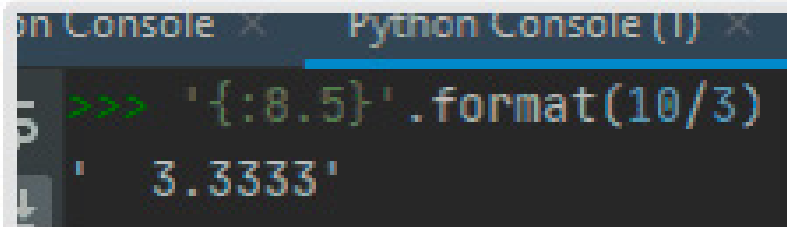
Largura de campo e função **format()**.

Observe que os valores 10 e 100 foram impressos com espaços em branco à esquerda. Isso ocorreu porque definimos que a primeira variável deveria ser impressa com 4 espaços com **{:4}** (2 foram ocupados e 2 ficaram em branco), e que a segunda variável deveria ser impressa com 5 espaços com **{:5}** (3 foram ocupados e 2 ficaram em branco).

Saiba mais

Também é válido perceber que o padrão é alinhar os valores à direita do espaço reservado para a impressão da variável.

O método **format()** também pode ser usado para imprimir valores de ponto flutuante com a precisão definida. Vamos a mais um exemplo:



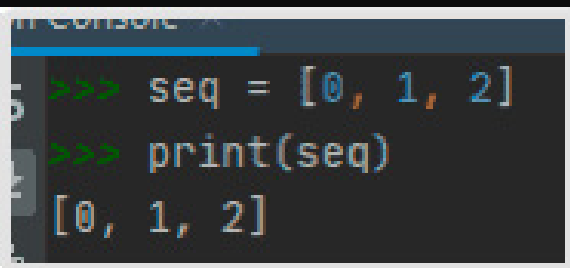
```
>>> '{:8.5}'.format(10/3)
'  3.3333'
```

Função format() com ponto flutuante.

Ao usar `{:8.5}`, estamos determinando que a impressão será com 8 espaços, mas apenas 5 serão utilizados.

Impressão de sequências

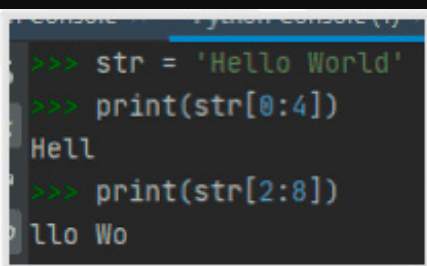
Python também permite a impressão de sequências com mais possibilidades que C, incluindo as strings. Para imprimir um vetor em C, por exemplo, precisamos chamar a `printf()` item a item. Em Python, basta chamar a função `print()` passando como parâmetro a sequência, veja:



```
>>> seq = [0, 1, 2]
>>> print(seq)
[0, 1, 2]
```

Função print() com sequência.

Para imprimir uma substring, por exemplo, basta utilizar os colchetes para indicar o intervalo de índices que deve ser impresso. Vale lembrar que o primeiro caractere da string é indexado com 0. Confira na imagem:



```
>>> str = 'Hello World'
>>> print(str[0:4])
Hell
>>> print(str[2:8])
llo Wo
```

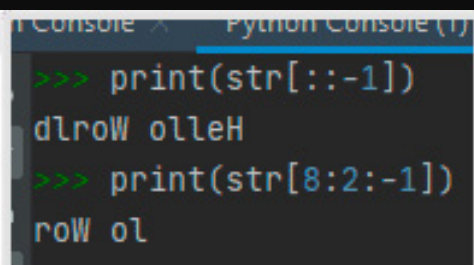
Função print() com substring.

Vale lembrar que:

Atenção!

Usar `[0:4]` provoca a impressão dos índices 0, 1, 2 e 3, mas não do índice 4. Analogamente, usar `[2:8]` provoca a impressão dos índices de 2 a 7, mas não do 8.

Também é possível imprimir a string como lida da direita para a esquerda. Para isso, deve-se utilizar `[: -1]`. Esse valor -1 indica que a leitura dos caracteres será feita no sentido oposto ao tradicional. Observe:



```
>>> print(str[::-1])
dlrow olleH
>>> print(str[8:2:-1])
row ol
```

Função `print()` – string em ordem reversa.

Vale lembrar que:

Atenção!

Fique atento quando utilizar o intervalo na impressão no sentido inverso, porque os limites do intervalo devem respeitar esse sentido.

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

(PGE-RO/2015/Técnico da Procuradoria/Tecnologia da Informação) Na linguagem Python, um comando como

Python



provoca:

A

A associação à variável "a" de uma função denominada "XXX" que pertence à biblioteca "input".

B

A criação de uma lista de valores denominada "a" cujo elemento inicial é a string "XXX".

C

A leitura de um valor do arquivo de entrada correntemente designado de acordo com um formato expresso pela string "XXX".

D

Um prompt no dispositivo de saída e a leitura de um valor que é armazenado na variável "a".

E

Cria na memória a variável XXX, que irá receber o valor digitado pelo usuário.

Parabéns! A alternativa D está correta.

A função `input()` tanto exibe na tela a string "XXX", como permite que o valor informado pelo usuário seja armazenado na variável `a`.

Questão 2

(TJ-BA/2015/Analista Judiciário/Tecnologia da Informação/Reaplicação) Analise o trecho de programa Python apresentado a seguir.

Python



Ao ser executado, o resultado exibido é:

A [1, 2, 3, 4, 5, 6, 7, 8]

B [8]

C []

D [8, 7, 6, 5, 4, 3, 2, 1]

E [1,3,5,7]

Parabéns! A alternativa D está correta.

A impressão da sequência L com a chamada L[::-1] é feita percorrendo toda a sequência L, em sentido inverso.

Considerações finais

Neste conteúdo você conheceu as principais características da linguagem Python e alguns conceitos relativos ao uso de variáveis, bem como aspectos concernentes à vinculação, como tempo e ambientes. Além disso, viu o conceito de escopo de visibilidade, tanto estático como dinâmico.

Nos módulos, foi feita uma referência a tipos de dados, como **int**, **float** ou **string**. Você estudou ainda as formas de atribuição, além de ter escrito seu primeiro programa em Python, utilizando o que aprendeu e, também, as formas de entrada e saída de dados que a linguagem oferece.

O uso correto desses conceitos é essencial na sua jornada de formação como programador. Recomendamos que fique atento aos detalhes e procure sempre programar de forma organizada. Isso vai evitar erros bobos e tornar sua experiência mais agradável.



Podcast

Ouçá agora um resumo dos principais tópicos aqui abordados sobre a linguagem Python.

Para ouvir o *áudio*, acesse a versão online deste conteúdo.



Referências

BELANI, G. **Programming Languages You Should Learn in 2020**. Consultado na internet em: 26 mai. 2020.

PERKOVIC, L. **Introdução à computação usando Python**: um foco no desenvolvimento de aplicações. Rio de Janeiro: LTC, 2016.

PYCHARM. **The Python IDE for Professional Developers**. Consultado na internet em: 15 jun. 2020.

PYTHON. **PEP 0 – Index of Python Enhancement Proposals (PEPs)**. Consultado na internet em: 2 jun. 2020.

PYTHON. **Fractions – Rational Number**. Consultado na internet em: 16 jun. 2020.

PYTHON. **Math – Mathematical Functions**. Consultado na internet em: 16 jun. 2020.

Explore +

Confira agora o que separamos especialmente para você!

Se você quiser ter mais exercícios para treinar e desafios mais complexos, recomendamos a visita ao site da Python Brasil.

Como vimos, dentre as PEPs, destaca-se a **PEP8**, que estabelece um guia de estilo de programação. Sugerimos que você pesquise mais sobre isso.

Para mais funções matemáticas, você pode utilizar os **módulos matemáticos math e fractions**.