Treinamento básico para programação em Python

Sumário

1 Algoritmos e paradigma de programação estruturada

Os algoritmos são as bases para criação de um programa de computador, onde diversas aplicações poderão ocorrer. Um algoritmo bem estruturado vai gerar um programa para solução de um problema que antes, parecia complexo.

A programação estruturada é uma forma de programação de computadores que estabelece uma disciplina de desenvolvimento de algoritmos, independentemente da sua complexidade e da linguagem de programação na qual será codificado, que facilita a compreensão da solução através de um número restrito de mecanismos de codificação.

2 O que é Python, para que serve e por que aprender?

As linguagens de programação possuem um papel fundamental tanto no desenvolvimento de sites e softwares, quanto em áreas mais complexas como ciência de dados, inteligência artificial, etc.

Neste conteúdo você irá aprender o que é Python, para que serve e por que aprender. Continue sua leitura e descubra o motivo dessa linguagem ser uma das mais populares entre os estudantes e profissionais da área de programação!

2.1 O que é Python?

Python é uma linguagem de programação de alto nível — ou High Level Language —, dinâmica, interpretada, modular, multiplataforma e orientada a objetos — uma forma específica de organizar softwares onde, a grosso modo, os procedimentos estão submetidos às classes, o que possibilita maior controle e estabilidade de códigos para projetos de grandes proporções.

Por ser uma linguagem de sintaxe relativamente simples e de fácil compreensão, ganhou popularidade entre profissionais da indústria tecnológica que não são especificamente programadores, como engenheiros, matemáticos, cientistas de dados, pesquisadores e outros.

Um de seus maiores atrativos é possuir um grande número de bibliotecas, nativas e de terceiros, tornando-a muito difundida e útil em uma grande variedade de setores dentro de desenvolvimento web, e também em áreas como análise de dados, machine learning e IA.

2.2 Qual a origem da linguagem Python?

Idealizada e desenvolvida por Guido Van Rossum, matemático holandês, no início dos anos 90, o Python foi criado com o objetivo de otimizar a leitura de códigos e estimular a produtividade de quem os cria, seja este um programador ou qualquer outro profissional.

A ideia surgiu, como quase todas as boas ideias, de uma necessidade: a de economizar tempo no desenvolvimento e melhorar a eficiência em um projeto desenvolvido no instituto onde Guido era pesquisador.

Para que esta melhoria pudesse ser feita de forma mais rápida e eficaz, Guido desenvolveu uma linguagem muito descomplicada e flexível: o Python.

Uma vez que esta linguagem passou a possibilitar a criação desde scripts muito simples até sistemas extremamente poderosos, profissionais de várias áreas começaram a progressivamente utilizá-la cada vez mais.

Hoje, além dos desenvolvedores de software, temos biólogos, contadores, físicos e outros profissionais potencializando suas habilidades através dela.

Estes são alguns motivos que têm feito o uso do Python crescer consideravelmente nos últimos anos em detrimento de outras linguagens.

2.3 Para que é usado o Python?

Como já citado acima, o Python é uma linguagem muito popular nas áreas da tecnologia relacionadas à análise de dados, pesquisa, desenvolvimento de algoritmos e IA. Mas, afinal, o que pode ser feito em Python?

2.3.1 Scripting e automação

Automatizar tarefas é um dos maiores objetivos de um profissional de tecnologia.

E desenvolver scripts de automação com Python é totalmente possível e rápido graças às suas várias bibliotecas nativas, disponíveis junto com sua instalação.

2.3.2 Desenvolvimento web

Uma das aplicações mais comuns do Python é o desenvolvimento de aplicações para web. Desde sites simples, como hotpages para ações de marketing, quanto ERPs capazes de administrar empresas e realizar vendas de produtos e serviços.

A linguagem possui uma extensa variedade de **frameworks** para todos os tipos de gosto, entre eles os famosos Django, Flask e FastAPI.

2.3.3 Enquadramento de testes

Se o momento de realizar testes era um momento desagradável no seu dia de trabalho, saiba que com o Python a sua vida será muito mais animada.

Ele possui um grande volume de estruturas de testes integradas, além de diversos módulos voltados exclusivamente para o enquadramento de testes.

Então, utilizar o Python é poder ter a certeza que sua aplicação poderá ser testada com muita eficiência.

Isso justifica por que desenvolvedores de software em Python adoram utilizar TDD (Test Driven Development), trazendo mais segurança e confiança aos seus projetos.

2.3.4 Big Data

A análise e interpretação de grandes volumes de dados, área de conhecimento que chamados de Big Data, é um dos mais populares campos onde Python tem se tornado a linguagem favorita.

Por ser facilmente compreendida por profissionais de diversas especialidades, o Python é, antes de tudo, um facilitador.

Além disso, duas características principais o levam a ser ótimo para essa tarefa: criação de scripts e a facilidade em utilizar as várias bibliotecas gratuitas desenvolvidas em parcerias com especialistas de várias áreas.

É possível analisar, processar e também exibir os dados com muita eficiência e clareza.

Embora ainda haja grandes concorrentes, como o R, por exemplo, o Python, assim como o próprio Big Data, está com a sua notoriedade no auge.

2.3.5 Ciência de dados

Acompanhando o próprio avanço da indústria tecnológica, o crescente número de bibliotecas disponíveis em Python voltadas à análise de dados oferece funções e métodos de otimização para praticamente quaisquer objetivos.

A grande comunidade de Python, não apenas internacional mas também no Brasil, é um poderoso atrativo: o compartilhamento de soluções e informações entre profissionais da área diminui consideravelmente as chances de um programador precisar lidar com um problema sem solução aparente.

2.3.6 Computação gráfica

Se você já assistiu aos mais recentes filmes da saga Star Wars, saiba que todos os seus efeitos de computação gráfica, produzidos pela Industrial Light & Magic, envolveu a linguagem Python.

Além de haver vários pacotes de soluções para esta área, como PyOpenGL e PyGame, há ainda o poderoso software de criação de gráficos 3D, o Blender, que utiliza o Python como linguagem principal.

2.3.7 Inteligência artificial

Quando você faz uma busca no Google e encontra exatamente a resposta que esperava, o Python desempenhou exatamente o papel que se esperava de sua atuação.

Isso porque os algoritmos e modelos de inteligência artificial presentes no buscador, elaborados para predizer a sua intenção de busca, são em sua maioria desenvolvidos nesta linguagem.

Entre as bibliotecas mais comuns voltadas ao aprendizado de máquina estão o TensorFlow, PyThorch, Theano, Keras e outras.

2.4 Quais as vantagens de programar em Python?

Como você deve estar percebendo, as vantagens de se dedicar ao aprendizado de Python são muitas, e entre elas está o fato de que os profissionais especializados nesta linguagem de programação no mercado são escassos.

Ou seja: ao se destacar como um bom programador Python, sua concorrência profissional será baixíssima.

Outros benefícios do Python são:

2.4.1 É fácil de aprender

A curva de aprendizado de um estudante de Python é, de modo geral, relativamente baixa. A linguagem, por ter uma sintaxe muito acessível e ter sido criada em prol da agilidade e da produtividade de quem a utiliza, é absorvida rápida e facilmente.

2.4.2 É portátil e multiplataforma

Por ser uma linguagem portátil e multiplataforma, o Python roda com tranquilidade em diversos sistemas operacionais, desde que seu interpretador esteja instalado. Além disso, o Python também é conhecido por suas propriedades extensíveis, tendo à sua disposição mais de 125.000 bibliotecas super versáteis.

2.4.3 É open source e gratuito

O Python é totalmente gratuito! Para instalar, utilizar e desenvolver em Python, basta simplesmente fazê-lo.

Além disso, a maior pesquisa realizada na área da programação, a StackOverflow Survey, perguntou este ano para desenvolvedores do mundo inteiro em qual linguagem eles mais gostam de programar e adivinhem: Python ficou em 1° lugar!

2.4.4 Oferece múltiplas possibilidades de desenvolvimento

É possível desenvolver diferentes tipos de aplicações com a linguagem Python.

As bibliotecas e frameworks disponíveis para essa linguagem ampliam a possibilidade de desenvolvimento. Um exemplo é o framework **Django**, que é usado para o desenvolvimento de aplicações web.

2.4.5 É uma linguagem “curinga”

A linguagem Python oferece diversas possibilidades no desenvolvimento de sistemas, ampliando desta forma as oportunidades no mercado de trabalho para os profissionais da área.

Ou seja, os programadores (as) poderão escolher entre diversos segmentos, além disso, sua sintaxe é simples e intuitiva, facilitando o aprendizado.

2.5 Quais empresas usam Python?

O Python é extremamente presente na vida de quem utiliza a internet com regularidade, principalmente por sua colaboração na criação de algoritmos, desde os menos complexos até funções de aprendizado de máquina.

Algumas empresas que utilizam Python e têm parte de seus serviços desenvolvidos nesta linguagem são Dropbox, Spotify, Airbnb e Uber.

As redes sociais Facebook, Instagram e Pinterest também têm algumas de suas funcionalidades escritas em Python.

Até mesmo a gigante NASA utiliza esta linguagem de programação!

Isso não significa, porém, que somente empresas de grande porte possam se beneficiar de seus benefícios competitivos.

2.6 Quanto tempo leva para aprender Python?

O Python é uma linguagem de programação de compreensão bastante acessível, com uma sintaxe simples e legibilidade clara, além de ter uma aprendizagem bastante rápida.

Para quem já tem alguma bagagem intelectual em **lógica de programação**, é possível aprender Python em apenas algumas semanas.

Há quem diga que, para os iniciantes, é possível tornar-se um programador de nível básico em apenas um mês, a depender da frequência de estudos.

De modo geral, a curva de aprendizagem de um aluno é muito individual e conectada a muitos aspectos de sua vida prática, como por exemplo tempo de dedicação e conhecimento prévio de disciplinas que podem estar relacionadas aos novos conceitos estudados.

2.7 Como anda o mercado para desenvolvedores Python?

Em uma única palavra: aquecido!

Segundo uma pesquisa realizada pela **Brasscom**, a busca por profissionais da área de TI é de 420 mil pessoas, até 2024, aqui no nosso país!

De acordo com eles, o Brasil forma 46 mil profissionais com perfil tecnológico por ano, ou seja, sobram vagas e falta mão de obra.

Atualmente, existe uma grande procura por Desenvolvedores Python, organizações de diferentes segmentos buscam esse profissional para ser um suporte dentro de alguns departamentos ou no setor de data science.

A maioria das vagas está dentro da **área de TI**, inteligência empresarial e marketing digital.

2.8 Empregos para quem estuda Python

Existem diversos caminhos para quem deseja seguir na carreira de programador(a) Python. Confira abaixo algumas opções para trabalhar usando essa linguagem.

2.8.1 Analista GIS

O software que descobre e analisa dados que podem ser vinculados a um determinado local é chamado de sistema de informações geográficas (GIS).

Um Analista GIS realiza a análise desses dados, mas esse profissional deve possuir mais conhecimento do que um técnico GIS, pois deve possuir as habilidades para utilizar um sistema de informação geográfica de maneira mais eficiente.

O curso de formação em Analista em Python com QGIS é recomendado para estudantes que queiram se especializar na programação Python em geoprocessamento e desenvolver ferramentas no software QGIS.

2.8.2 Desenvolvedor de software

As empresas estão buscando cada vez mais programadores em Python para atuar em distintas áreas como: Inteligência Artificial, análise de dados, computação gráfica, big data, automação e desenvolvimento de dados.

Confira abaixo algumas áreas de atuação para um desenvolvedor em Python:

- Indústria de jogos;
- Testes web;
- Administração de sistemas;
- Programação;
- Mobile;
- Protótipos de Sistemas.

Existe uma grande falta de programadores em Python no mercado de trabalho, o que acaba sendo refletido diretamente nos salários pagos.

2.8.3 Engenheiro de Qualidade

O engenheiro da qualidade é responsável por gerenciar e manter todos os detalhes e processos relacionados à qualidade de produtos e serviços. Baseando-se em diversas estratégias, os profissionais dessa área trabalham para assegurar que as expectativas do consumidor sejam atendidas.

O engenheiro de qualidade, também é o responsável por identificar erros e possíveis melhorias, para isso o profissional aplica várias ferramentas e metodologias.

Algumas das principais funções desses engenheiros de qualidade são:

- Desenvolvimento de planos estratégicos para implementação de metodologias de Engenharia da Qualidade;
- Gestão de qualidade através de Sistemas de Gestão da Qualidade (SGQ);
- Utilização de recursos como o Quality Function Deployment (QFD) e Análise de Modos e Efeitos de Falha (FMEA);
- Aplicação do VOC (Vision of Customer ou Voz do cliente) no desenvolvimento de novos produtos ou na otimização de produtos já existentes;
- Identificação de erros, falhas, desperdícios em processos de produção e aplicação de estratégias de solução;
- Desenvolvimento e aplicação de testes e inspeções;
- Gestão e treinamento de equipes com o intuito de estabelecer ou melhorar processos de Engenharia da Qualidade.

2.8.4 Desenvolvedor Full Stack

Como sua própria denominação indica, o **desenvolvedor full stack** é o profissional habilitado para compreender e operar em todas as camadas do desenvolvimento de um projeto, desde a criação de servidores internos até interfaces de comunicação com o usuário final.

Confira com mais detalhes o que significa cada uma destas frentes.

2.8.5 Desenvolvimento Front-end

O **desenvolvimento front-end** diz respeito a toda parte da frente de uma aplicação. Ou seja, todo fragmento de um site, software ou aplicativo com o qual o usuário tem contato direto e pode ser visto pelo computador ou através de qualquer outra tela.

A disposição das páginas de um site, sua aparência, layout, aplicações de interação, enfim, tudo aquilo com o que o usuário pode se comunicar está dentro do escopo de trabalho de um desenvolvedor front-end.

O desenvolvimento front-end também é chamado de client-side.

As mais populares linguagens de programação voltadas ao front-end são React e JavaScript.

Também faz parte das habilidades exigidas deste programador que ele saiba trabalhar com variados tipos de frameworks e bibliotecas. Entre elas, as mais comuns são jQuery, VueJS, Angular, TailwindCSS e Bulma.io.

2.8.6 Desenvolvimento Back-end

Como antagonista ao front-end, desenvolvimento back-end é tudo aquilo que envolve o core de uma aplicação, ou seja, tudo aquilo que o usuário não vê e que diz respeito à sua infraestrutura interna e funcionamento.

O **desenvolvimento back-end** também é chamado de server-side.

As tecnologias mais comuns no desenvolvendo back-end são Java, C#, PHP, Node.js, Ruby, Python, etc.

Entre os frameworks mais utilizados por esses programadores estão Django, Rails, Laravel, Phoenix e Spring Boot.

Embora existam profissionais especialistas em banco de dados, algum conhecimento para trabalhar com os sistemas mais comuns de database, como MySQL, PostreSQL, MondoDB, Cassandra, ElasticSearch e Redis são bastante convenientes.

Uma vez que o desenvolvedor full stack está capacitado para agir tanto em front como em back-end, este tipo de profissional é comumente contratado para a liderança de projetos de grandes proporções.

2.8.7 Engenheiro de Aprendizado de Máquina

O machine learning, ou aprendizado de máquinas em português, é um ramo da Inteligência Artificial (AI). O profissional dessa área possui o objetivo de ensinar as máquinas a realizar tarefas imitando o raciocínio humano.

O Engenheiro de Aprendizado de Máquina precisa compreender qual a situação do negócio, e qual o problema ele espera que a Inteligência Artificial ajude a resolver.

Depois disso começam as etapas de desenvolvimento de modelos de Machine Learning, desde as fases de pesquisa e estudo, passando por protótipos até a criação, manutenção e treinamento desses sistemas.

Esse profissional fica muito próximo dos demais membros da equipe de tecnologia, como os Cientistas e Engenheiros de Dados.

3 Operadores em Python

Os Operadores em Python possibilitam que o desenvolvedor consiga transcrever a **lógica** para **código**.

Python disponibiliza uma série desses operadores para os desenvolvedores e é muito importante dominá-los se você quiser se tornar um **verdadeiro Pythonista**!

Veremos todos em detalhes agora, começando pelos **Operadores Aritméticos**!

3.1 Operadores Aritméticos

Esses operadores são utilizados para criarmos expressões matemáticas comuns, como soma, subtração, multiplicação e divisão.

Veja quais estão disponíveis no Python:

Operador	Nome	Função
+	**Adição**	Realiza a soma de ambos operandos.
-	**Subtração**	Realiza a subtração de ambos operandos.
*	**Multiplicação**	Realiza a multiplicação de ambos operandos.
/	**Divisão**	Realiza a Divisão de ambos operandos.
//	**Divisão inteira**	Realiza a divisão entre operandos e a parte decimal de ambos operandos.
%	**Módulo**	Retorna o resto da divisão de ambos operandos.
**	**Exponenciação**	Retorna o resultado da elevação da potência pelo outro.

Veja agora a utilização de cada operador aritmético mencionado acima:

```
1  quatro = 4
2  dois = 2
3
4  soma = quatro + dois
5  print(soma)  # Resultado: 6
6
7  subtracao = quatro - dois
8  print(subtracao)  # Resultado: 2
9
10 multiplicacao = quatro * dois
11 print(multiplicacao)  # Resultado: 8
12
13 divisao = quatro / dois
14 print(divisao)  # Resultado: 2.0
15
16 divisao_interna = quatro // dois
17 print(divisao_interna)  # Resultado: 2
18
19 modulo = quatro % dois
20 print(modulo)  # Resultado: 0
21
22 exponenciacao = quatro ** dois
23 print(exponenciacao)  # Resultado: 16
```

3.2 Operadores relacionais

Como o nome já diz, esses Operadores são usados para **comparar** dois valores:

Operador	Nome	Função
==	Igual a	Verifica se um valor é igual ao outro
!=	Diferente de	Verifica se um valor é diferente ao outro
>	Maior que	Verifica se um valor é maior que outro
>=	Maior ou igual	Verifica se um valor é maior ou igual ao outro
<	Menor que	Verifica se um valor é menor que outro
<=	Menor ou igual	Verifica se um valor é menor ou igual ao outro

Vamos ver exemplos da utilização de cada operador de comparação mencionado acima.

Para facilitar o entendimento, todas as operações estão retornando um valor igual a True, para que você entenda como cada condição é aceita:

```
1   var = 5
2
3   if var == 5:
4       print('Os valores são iguais')
5
6   if var != 7:
7       print('O valor não é igual a 7')
8
9   if var > 2:
10      print('O valor da variável é maior de 2')
11
12  if var >= 5:
13      print('O valor da variável é maior ou igual a 5')
14
15  if var < 7:
16      print('O valor da variável é menor que 7')
17
18  if var <= 5:
19      print('O valor da variável é menor ou igual a 5')
```

Resultado do código acima:

```
Os valores são iguais
O valor não é igual a 5
O valor da variável é maior de 5
O valor da variável é maior ou igual a 5
```

```
O valor da variável é menor que 7
O valor da variável é menor ou igual a 5
```

3.3 Operadores de Atribuição

Esse Operadores são utilizados no momento da **atribuição** de valores à variáveis e controlam como a atribuição será realizada.

Veja quais Operadores de Atribuição estão disponíveis em Python:

Operador	Equivalente a
=	x = 1
+=	x = x + 1
-=	x = x - 1
*=	x = x * 1
/=	x = x / 1
%=	x = x % 1

Exemplo da utilização de cada operador de atribuição mencionado acima:

👉 **Operador `+=`:**

```
1  numero = 3
2
3  numero += 7
4  print(numero)  # Resultado será 10
```

👉 **Operador `-=`:**

```
1  numero = 5
2
3  numero -= 3
4  print(numero)  # Resultado será 2
```

👉 **Operador *=**:

```
numero = 5

numero *= 2
print(numero)  # Resultado será 10
```

👉 **Operador /=**:

```
numero = 5

numero /= 4
print(numero)  # Resultado será 1.25
```

👉 **Operador %=**:

```
numero = 5

numero %= 2
print(numero)  # Resultado será 1
```

Obs: O operador % é chamado módulo e nada mais é que o resto da divisão. No exemplo acima: 5 dividido por 2 dá 2 de resultado e sobra 1. Por isso numero %= 2 será 1!

3.4 Operadores Lógicos

Esses Operadores nos possibilitam construir um tipo de teste muito útil e muito utilizado em qualquer programa Python: os **testes lógicos**.

Python nos disponibiliza três tipos de Operadores Lógicos: o and, o or e o not.

Vamos ver mais sobre eles agora!

Operador	Definição
and	Retorna True se ambas as afirmações forem verdadeiras
or	Retorna True se uma das afirmações for verdadeira
not	retorna Falso se o resultado for verdadeiro

Exemplo da utilização de cada um:

```
num1 = 7
num2 = 4

# Exemplo and
if num1 > 3 and num2 < 8:
    print("As Duas condições são verdadeiras")

# Exemplo or
if num1 > 4 or num2 <= 8:
    print("Uma ou duas das condições são verdadeiras")

# Exemplo not
if not (num1 < 30 and num2 < 8):
    print('Inverte o resultado da condição entre os parânteses')
```

3.5 Operadores de Identidade

Estes Operadores são utilizados para *comparar* objetos, verificando se os objetos testados referenciam o mesmo objeto (is) ou não (is not).

Operador	Definição
is	Retorna True se ambas as variáveis são o mesmo objeto
is not	Retorna True se ambas as variáveis não forem o mesmo objeto

Agora vamos aos exemplos de como utilizar cada operador de identidade mencionado acima:

Exemplo do operador is:

```
lista = [1, 2, 3]
outra_lista = [1, 2, 3]
recebe_lista = lista

# Recebe True, pois são o mesmo objeto
print(f"São o mesmo objeto? {lista is recebe_lista}")

# Retorna False, pois são objetos diferentes
print(f"São o mesmo objeto? {lista is outra_lista}")
```

Resultado do código acima:

```
São o mesmo objeto? True
São o mesmo objeto? False
```

Exemplo do operador `is not`:

```
tupla = (1, 2, 3)
outra_tupla = (1, 2, 3)

print(f"Os objetos são diferentes? {outra_tupla is tupla}")
```

Resultado do código acima:

```
Os objetos são diferentes? True
```

Muitas vezes programadores Python ficam na dúvida em quando utilizar o operador de igualdade `==` ou o operador de identidade `is`.

Mas agora que você já conhece os dois sabe que o operador `==` verifica os **valores** testados, enquanto o operador `is` testa a **referência** dos valores testados! 😉

3.6 Operadores de Associação

Por último, temos os Operadores de Associação.

Eles servem para verificar se determinado objeto está **associado** ou **pertence** a determinada estrutura de dados.

Operador	Função
`in`	Retorna `True` caso o valor seja encontrado na sequência
`not in`	Retorna `True` caso o valor não seja encontrado na sequência

Exemplo da utilização de cada operador de associação mencionado acima:

```
lista = ["Python", 'Academy', "Operadores", 'Condições']

# Verifica se existe a string dentro da lista
print('Python' in lista)  # Saída: True

# Verifica se não existe a string dentro da lista
print('SQL' not in lista) # Saída: True
```

4 Variáveis e constantes

4.1 Regras de nomeação de variáveis

A linguagem de programação Python possui como regra de nomeação de variáveis o formato snake_case.

Este formato determina que o nome das variáveis possuam um padrão em que, as palavras sejam definidas em letras minúsculas e, caso possua, os espaços serão substituídos por “underline” conforme podemos verificar abaixo:

```
variavel_com_nome_composto = "Treinaweb"

print(variavel_com_nome_composto) # Treinaweb
```

4.2 Declarando variáveis

A declaração de variáveis no Python é um processo muito simples. Por não possuir um escopo padrão para seus scripts, basta definir o nome da variável e, em seguida, atribuir o valor desejável:

```
nome_da_variavel = "valor_da_variavel"
```

4.3 Declarando constantes

A regra de nomeação das constantes no Python segue um padrão parecido com as de variáveis, com a diferença de que todas as letras são maiúsculas e separadas por underline “_”. Porém, por possuir tipagem dinâmica, os valores atribuídos à constantes podem ser alterados sem problemas:

```
MINHA_CONSTANTE = 10

print(MINHA_CONSTANTE) # 10

MINHA_CONSTANTE = 15

print(MINHA_CONSTANTE) # 10
```

4.4 Tipos de dados

4.4.1 Tipo Inteiro (int)

O tipo inteiro é um tipo composto por caracteres numéricos (algarismos) inteiros.

É um tipo usado para um número que pode ser escrito sem um componente decimal, podendo ter ou não sinal, isto é: ser positivo ou negativo.

Por exemplo, 21, 4, 0, e −2048 são números inteiros, enquanto 9.75, 1/2, 1.5 não são.

Exemplos:

```
1  idade = 18
2  ano = 2002
3
4  print(type(idade))
5  print(type(ano))
```

Saída:

```
<class 'int'>
<class 'int'>
```

4.4.2 Ponto Flutuante ou Decimal (float)

É um tipo composto por caracteres numéricos (algarismo) decimais.

O famoso ponto flutuante é um tipo usado para números racionais (números que podem ser representados por uma fração) informalmente conhecido como “número quebrado”.

Exemplos:

```
1  altura = 1.80
2  peso = 73.55
3
4  print(type(peso))
5  print(type(altura))
```

Saída:

```
<class 'float'>
<class 'float'>
```

4.4.3 String (str)

É um conjunto de caracteres dispostos numa determinada ordem, geralmente utilizada para representar palavras, frases ou textos.

Exemplos:

```
1  nome = “Guilherme”
2  profissao = 'Engenheiro de Software'
3
4  print(type(profissao))
5  print(type(nome))
```

Saída:

```
<class 'str'>
<class 'str'>
```

4.4.4 Boolean (bool)

Tipo de dado lógico que pode assumir apenas dois valores: falso ou verdadeiro (`False` ou `True` em Python).

Na lógica computacional, podem ser considerados como 0 ou 1.

Exemplos:

```
1 fim_de_semana = True
2 feriado = False
3
4 print(type(fim_de_semana))
5 print(type(feriado))
```

Saída:

```
<class 'bool'>
<class 'bool'>
```

4.4.5 Listas (list)

Tipo de dado muito importante e que é **muito** utilizado no dia a dia do desenvolvedor Python!

Listas agrupam um conjunto de elementos variados, podendo conter: inteiros, floats, strings, outras listas e outros tipos.

Elas são definidas utilizando-se colchetes para delimitar a lista e vírgulas para separar os elementos, veja alguns exemplo abaixo:

```
1 alunos = ['Amanda', 'Ana', 'Bruno', 'João']
2 notas = [10, 8.5, 7.8, 8.0]
3
4 print(type(alunos))
5 print(type(notas))
```

Saída:

```
<class 'list'>
<class 'list'>
```

4.4.6 Tuplas (tuple)

Assim como Lista, Tupla é um tipo que agrupa um conjunto de elementos.

Porém sua forma de definição é diferente: utilizamos parênteses e também separado por vírgula.

A diferença para Lista é que Tuplas são **imutáveis**, ou seja, após sua definição, Tuplas **não podem ser modificadas**.

Vamos ver alguns exemplos:

```
1  valores = (90, 79, 54, 32, 21)
2  pontos = (100, 94.05, 86.8, 62)
3
4  print(type(valores))
5  print(type(pontos))
```

Saída:

```
<class 'tuple'>
<class 'tuple'>
```

Caso haja uma tentativa de alterar os itens de uma tupla após sua definição, como no código a seguir:

```
1  tuple = (0, 1, 2, 3)
2  tuple[0] = 4
```

O seguinte erro do tipo TypeError será lançado pelo interpretador do Python:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

4.4.7 Dicionários (dict)

Dict é um tipo de dado muito flexível do Python.

Eles são utilizados para agrupar elementos através da estrutura de chave e valor, onde a chave é o primeiro elemento seguido por dois pontos e pelo valor.

Vamos ver alguns exemplos:

```
1  altura = {'Amanda': 1.65, 'Ana': 1.60, 'João': 1.70}
2  peso = {'Amanda': 60, 'Ana': 58, 'João': 68}
3
4  print(type(altura))
5  print(type(peso))
```

Saída:

```
<class 'dict'>
```

```
<class 'dict'>
```

4.4.8 Como mudar o tipo de uma variável

Em determinados cenários pode ser necessário mudar o tipo de uma variável e no Python isso é muito fácil, uma das vantagens de uma linguagem dinamicamente tipada.

Abaixo veremos exemplos de como trocar o tipo de variáveis.

4.4.8.1 Decimal (float) para String (str)

```
1  # Antes da conversão
2  altura = 1.80
3  print(type(altura))
4
5  # Conversão do tipo
6  altura = str(altura)
7
8  # Depois da conversão
9  type(altura)
10 print(altura)
```

Saída:

```
<class 'float'>
<class 'str'>
'1.8'
```

4.4.8.2 Inteiro (int) para Decimal (float):

```
1  # Antes da conversão
2  idade = 18
3  print(type(idade))
4
5  # Conversão do tipo
6  idade = float(idade)
7
8  # Depois da conversão
9  print(type(idade))
10 print(idade)
```

Saída:

```
<class 'int'>
<class 'float'>
18.0
```

4.4.8.3 Booleano (bool) para Inteiro (int)

```
1  # Antes da conversão
2  fim_de_semana = True
3  print(type(fim_de_semana))
4
```

```
5   # Conversão do tipo
6   fim_de_semana = int(fim_de_semana)
7
8   # Depois da conversão
9   print(type(fim_de_semana))
10  print(fim_de_semana)
```

Saída:

```
<class 'bool'>
<class 'int'>
1
```

5 Estruturas de seleção

if/else em Python

Quando programamos, muitas vezes precisamos que determinado bloco de código seja executado apenas se uma determinada condição for verdadeira. Em casos assim, devemos fazer uso de uma estrutura de condição.

Neste documento você aprenderá a utilizar a estrutura de condição `if-else` **e** `elif` em Python.

5.1 if

O **if** é uma estrutura de condição que permite avaliar uma expressão e, de acordo com seu resultado, executar uma determinada ação.

No código a seguir temos um exemplo de uso do **if** no qual verificamos se a variável idade é menor que 20. Em caso positivo, imprimimos uma mensagem na tela e em caso negativo o código seguirá normalmente, desconsiderando a linha 3.

```
idade = 18

if idade < 20:

    print('Você é jovem!')
```

Como podemos notar, essa estrutura é formada pela palavra reservada **if**, seguida por uma condição e por dois pontos (:). As linhas abaixo dela formam o bloco de instruções que serão executadas se a condição for atendida. Para isso, elas devem ser identadas corretamente, respeitando a **especificação do Python**. Nesse código, apenas a instrução da linha 3 é executada, e por isso ela está tabulada. Se fosse necessária a execução de outras linhas no caso da idade ser menor que 20, elas também deveriam estar no mesmo nível de identação da linha 3.

5.2 if-else

Vimos anteriormente como utilizar o if para executar uma ação caso uma condição seja atendida. No entanto, nenhum comportamento específico foi definido para o caso de a condição não ser satisfeita. Quando isso é necessário, precisamos utilizar a reservada **else**.

```
idade = 18

if idade >= 18:

    print('maior de idade')

else:

    print('menor de idade')
```

Dessa vez, caso a condição avaliada na linha 3 não seja atendida, definimos o fluxo alternativo que o código deve seguir. Ou seja, se a idade não for maior ou igual a 18, o bloco abaixo da palavra reservada else deverá ser executado. Nesse caso, temos apenas uma instrução de impressão (linha 5).

5.3 if-elif-else

Adicionalmente, se existir mais de uma condição alternativa que precisa ser verificada, devemos utilizar o **elif** para avaliar as expressões intermediárias antes de usar o **else**, da seguinte forma:

```
idade = 18

if idade < 12:

    print('crianca')

elif idade < 18:

    print('adolescente')

elif idade < 60:

    print('adulto')

else:

    print('idoso')
```

Na linha 2 definimos a primeira condição (idade < 12). Caso essa não seja atendida, o programa seguirá para a linha 4 e avaliará a próxima condição (elif), que se for verdadeira fará com que o bloco logo abaixo (a linha 5, nesse caso) seja executado. Caso essa condição ainda não seja atendida (elif), há uma outra alternativa na linha 6 que será avaliada e que fará com que o bloco logo abaixo seja executado se ela for atendida. Por fim, se nenhuma das condições for satisfeita, o programa seguirá para a linha 8, executando o que é definido pelo else.

6 Estruturas de repetição

Veremos como funcionam e como utilizar as estruturas de controle por repetição em Python: **FOR** e **WHILE**.

Em algumas situações, é comum que uma mesma instrução (ou conjunto delas) precise ser executada várias vezes seguidas. Nesses casos, normalmente utilizamos um loop (ou laço de repetição), que permite executar um bloco de código repetidas vezes, enquanto uma dada condição é atendida.

Em Python, os loops são codificados por meio dos comandos for e while. O primeiro nos permite percorrer os itens de uma coleção e, para cada um deles, executar um bloco de código. Já o while, executa um conjunto de instruções várias vezes enquanto uma condição é atendida.

6.1 for

```
nomes = ['Pedro', 'João', 'Leticia']

for n in nomes:

    print(n)
```

A variável definida na linha 1 é uma lista inicializada com uma sequência de valores do tipo string. A instrução for percorre todos esses elementos, um por vez e, em cada caso, atribui o valor do item à variável n, que é impressa em seguida. O resultado, então, é a impressão de todos os nomes contidos na lista.

6.2 while

O comando while, por sua vez, faz com que um conjunto de instruções seja executado enquanto uma condição for atendida. Quando o resultado passa a ser falso, a execução é interrompida, saindo do loop, e passa para o próximo bloco.

```
contador = 0

while contador < 5:

    print(contador)

    contador = contador + 1
```

Observe que na linha 4 incrementamos a variável contador, de forma que em algum momento seu valor igual a 5. Quando isso for verificado na linha 2, o laço será interrompido. Caso a condição de parada nunca seja atingida, o loop será executado infinitamente, gerando problemas no programa.

Estruturas de repetição estão presentes na maioria das linguagens de programação e representam uma parte fundamental de cada uma delas. Sendo assim, é muito importante conhecer a sintaxe e o funcionamento dessas estruturas.

7 Exercícios

01. Criar um programa em Python que receba um valor de temperatura em Celsius, calcule e imprima no console o valor equivalente em Fahrenheit.

Fórmula: https://tecnoblog.net/responde/como-converter-graus-celsius-para-fahrenheit/#:~:text=Ou%20seja%2C%20para%20que%20você,1%2C8%20e%20somar%2032.

02. Criar um programa em Python que calcule regra de 3 simples.

Fórmula: https://www.todamateria.com.br/regra-de-tres-simples-e-composta/

03. Criar um programa em Python que receba um valor em horas, calcule e imprima no console o valor equivalente em minutos.

04. Criar um programa em Python que recebe o valor de um produto e o valor recebido em dinheiro, e imprima no console o valor do troco que deve ser devolvido ao comprador.

05. Criar um programa em Python que recebe o valor da idade de uma pessoa e informe se o voto é obrigatório, ou seja, imprima True caso ela tenha entre 18 e 60 anos, ou False caso contrário.

06. Criar um programa em Python que recebe um valor em reais e converta para pesos, sendo que 1 real custa 1.6 pesos. Imprimir o resultado e imprimir True caso o valor seja maior do que 100.

07. Criar um programa em Python que calcule o IMC, índice de massa corporal, e imprima True caso o índice seja maior do que 25.0 e menor do que 35.0; Fórmula: IMC = Peso (Kg)/ (Altura(m))^2

08. Criar um programa em Python que calcula o gasto de combustível para uma viagem. O usuário precisa informar qual é a distancia que percorrerá, o custo atual do combustível e a média de consumo do seu veículo. O programa deve imprimir o valor em reais no console.

09. Criar um programa em Python que receba o valor do salário de um trabalhador, calcule e imprima na tela o valor do seu novo salário após reajuste de 20%.

10. Criar um programa em Python que calcule a fórmula $E = mc^2$, na qual E representa a energia, m representa a massa e c, representa a velocidade da luz no vácuo. Lembrando que o valor da velocidade de luz pode ser considerado fixo em $3x10^8$, o usuário deve informar apenas o valor da massa.

Obs.: implemente cada exercício em um arquivo diferente, e utilize variáveis com nomes expressivos.

Exercícios de estruturas de seleção de estruturas de repetição:

01. Crie um programa que recebe um número e imprime na tela se ele é negativo, positivo ou igual a zero.

02. Criar um programa no qual o usuário informa a quantidade de maçãs que deseja comprar e imprima ao final o custo total, considerando que se ele comprar até 10 unidades cada maçã custará R$ 0.50, entre 10 e 20 unidades custará R$ 0.40 e acima de 20 unidades custará R$ 0.30

03. Criar um programa que receba 3 valores inteiros e imprima na tela na ordem crescente.

04. Escreva um programa que receba o tamanho de cada um dos tres lados de um triangulo e imprima na tela se ele é Equilátero, Isóceles ou Escaleno.

05. Escrever um programa que recebe o valor dos 3 angulos de um triangulo e imprima na tela se ele é Acutângulo, Retângulo ou Obtusângulo.

06. Criar um programa para calcular a fórmula de báskara.

07. Crie um programa que recebe o valor do raio de uma circunferencia e imprima na tela uma mensagem caso a área seja supeior do que 10 e outra mensagem caso o perimetro seja superior que 10.

08. Criar um algoritmo que calcule o fatorial de um número.

09. Criar um algoritmo que calcule a sequencia de fibonacci até a N-ésima iteração.

10. Criar um programa que receba horário do relógio, considerando horas (H) e minutos (M) e calcule qual é o angulo entre os ponteiros do relógio.

11. Criar um script que deve receber um número N e imprimir na tela todos os números inteiros entre 0 e N.

22. Criar um script que deve receber um número N e imprimir na tela todos os números inteiros entre 0 e N de forma decrescente.

13. Criar um script que deve receber um número N e imprimir na tela todos os números inteiros pares entre 0 e N.

14. Criar um programa que recebe um número N e imprime sua tabuada.

15. Escreva um script que conte quantos números entre 0 e 100 são múltiplos de 5.

16. Criar um script que receba 10 números e calcule a média entre eles.

17. Criar um script para ler 10 números e ao final imŕimir quantos são pares e quantos são ímpares.

18. Criar um programa que receba 10 números e conte quantos deles são negativos e quantos são positivos.

19. Criar um algoritmo que receba 10 números e ao final imprima na tela o menor e o maior deles.

20. Criar um algoritmo que respoda se determinado número é um número primo.

21. Chico tem 1,50 e cresce 2 centímetros por ano, enquanto Juca tem 1,10 e cresce 3 centímetros por ano. Construir um programa que calcule e imprima quanto anos seriam necessários para que Juca passe a ser maior que Chico.

22. Elabore um programa que solicite dois valores inteiros A e B e efetue a multiplicação destes valores e apresente o resultado. Entretanto, para elaboração deste programa só poderão ser utilizados os operadores aritméticos de soma e subtração, ou seja, NÃO poderá ser utilizado ooperador de multiplicação.