Mini Curso Python 3

Victor Luz

O meu nome é Victor Luz e carrego Python comigo à décadas, ou Python carrega comigo à décadas.

Uma das duas, ou talvez ambas. Não sei.

1989

```
◎ Protestos na Praça Tiananmen
◎ Lançamento da nave espacial Galileu pela NASA
◎ A Guerra Fria termina na sequência da Conferência de Malta
◎ O Muro de Berlim é derrubado
◎ Exxon Valdez derrama 11 milhões de galões de petróleo
◎ A Nintendo lança o Game Boy
◎ Eleições livres na Polónia levam o Solidariedade ao poder
```

De acordo com a narrativa tradicional:

Estávamos no Natal de 1989, a cidade é Amesterdão nos Países Baixos.

Guido van Rossum estava sentado em casa a pensar como passar uma semana de férias enquanto o escritório estava fechado. Queria algo interessante para o manter ocupado.

Nessa altura, estava a trabalhar no projeto Amoeba. Era um sistema distribuído baseado em microkernel para o qual ele estava a desenvolver utilitários de sistema.

Durante o trabalho neste projeto, van Rossum apercebeu-se que codificar em C demorava muito tempo. Ele queria uma linguagem de script, mais fácil e legível do que os scripts de shell da altura.

Com o tempo livre nesse Natal, van Rossum começou a desenvolver a ideia.

De imediato, pensou no nome "Python" para o seu novo projeto. Era fã da trupe britânica de comédia Monty Python. Este era um nome adequadamente irreverente para o que era essencialmente um "projeto de sucata".

O nome era curto, cativante, um pouco ousado, e em consonância com a tradição de dar nomes famosos às linguagens de programação.

Ele queria que esta nova linguagem fosse clara e fácil de aprender. Baseia-se numa utilização simplificada da língua inglesa e código-fonte aberto.

A filosofia do Python está resumida em "O ZEN de Python".

```
--> import this
```

1. Bonito é melhor do que feio.

Um código bonito é melhor do que um código feio. Se dois trechos de código estiverem ambos a funcionar, mas um é simples e facilmente legível enquanto o outro é difícil de entender, o primeiro é definitivamente o vencedor.

2. Explícito é melhor do que implícito.

O código deve ser compreensível para alguém que não sabe nada acerca do seu programa. Nenhum conhecimento prévio deve ser exigido.

3. Simples é melhor do que complexo.

Se tiver um problema simples que possa ser resolvido com uma solução simples, avance. Se tiveres um problema complexo, divide-o em vários problemas que podem ser resolvidos com uma solução simples.

4. Complexo é melhor que complicado.

Quando o problema requer uma solução complexa, ela não deve ser demasiado complicada. Código complicado deixa seus colegas confusos e consome toneladas de tempo e esforço para entender.

5. Liso é melhor do que agrupado.

É um fã de organizar as coisas em categorias, subcategorias e sub-subcategorias? Quando se trata de organizar código, essa estrutura hierárquicas adicionam mais confusão do que organização.

6. Esparso é melhor que denso.

Normalmente, é preferível ter várias linhas de código fáceis de seguir do que uma linha única e densa.

7. A legibilidade conta.

Muitas vezes, os programadores tentam poupar tempo abreviando nomes de variáveis e funções ou saltando comentários. Lembre-se que pode escrever código apenas uma vez, mas outros terão de o ler várias vezes. Se quiser realmente poupar tempo, torne o seu código legível usando nomes de variáveis e funções fáceis de entender, documentação extensa e indentação correta.

8. Casos especiais não são suficientemente especiais para quebrar as regras.

Em Python, existem boas práticas que tornam o teu código mais legível para outros programadores. Siga estas práticas em vez de fazer "à sua maneira". Esta regra é especialmente importante quando cria módulos ou bibliotecas para outros usarem.

9. Embora a prática vença a pureza.

Cada regra pode ter uma exceção. Se for mais prático resolver um problema "à sua maneira" e isso mantém o código legível, fácil de seguir, pode por vezes desviar-se das melhores práticas estabelecidas. Pode ser um desafio para os novos programadores navegar entre este e o princípio acima, mas fica mais fácil com a experiência.

10. Os erros nunca devem passar silenciosamente.

Se houver um erro no seu programa, retornar nada ou apenas um código de erro, está a ter um erro silencioso. Isso não é bom. Silenciar erros eventualmente leva a bugs que são mais difíceis de eliminar. É melhor que um programa falhe do que silenciar um erro e continuar a correr.

11. A menos que explicitamente silenciado.

Em alguns casos, você pode querer ignorar os erros que seu programa pode causar. Então a melhor prática é silenciar esse erro explicitamente no seu código.

12. Perante a ambiguidade, recuse a tentação de adivinhar.

Se o seu código não estiver a funcionar, não tente cegamente soluções diferentes até que uma delas pareça funcionar. Poderá estar apenas a mascarar o problema em vez de o resolver. Em vez disso, aplique pensamento crítico para para compreender o problema e encontrar uma solução adequada.

13. Deve haver uma - e de preferência apenas uma - forma óbvia de o fazer.

Existe flexibilidade quando se tem várias soluções para o mesmo problema. No entanto, isso também aumenta a complexidade e o esforço, pois é necessário estar familiarizado com todas as soluções possíveis.

14. Embora essa forma possa não ser óbvia à primeira vista, a menos que sejas holandês.

Este princípio refere-se ao criador do Python, Guido van Rossum, que é Holandês. Obviamente, recordar e compreender qualquer regra em Python seria mais fácil para ele do que para qualquer outra pessoa.

15. Agora é melhor do que nunca.

Este princípio tem pelo menos duas interpretações diferentes. Algumas pessoas pensam que ele se refere a loops infinitos e intermináveis que você, obviamente, deve evitar no seu código. Outra interpretação possível é que só precisa de evitar a procrastinação nos seus projetos de

Não faz mal algum aceitar ambas as interpretações.

16. Embora nunca seja frequentemente melhor do que agora.

Este princípio desenvolve as ideias do anterior. Se o aplicarmos diretamente ao nosso código, podemos dizer que é melhor esperar que um programa termine do que terminá-lo cedo e obter resultados incorretos. Se estivermos a falar de projetos de programação em geral, podemos interpretar este princípio como um apelo para pensar e planear o projeto em vez de nos lançarmos de cabeça.

17. Se a implementação for difícil de explicar, é uma má ideia.

Se for difícil para si explicar a sua implementação aos seus colegas, é provável que tenha escrito um código mau. O mais provável é que o tenha feito muito complicado e violou alguns dos princípios acima. Tente simplificar sua solução até que você se sinta confortável em explicá-la a outros.

18. Se a implementação for fácil de explicar, pode ser uma boa ideia.

Se conseguir explicar facilmente a sua implementação aos seus colegas, pode ser uma boa ideia. Pode ainda estar errada, mas está no bom caminho em termos de legibilidade e simplicidade do código.

19. Namespaces são uma boa ideia.

Em Python, podes ter espaços de nomes isolados ou uma coleção de nomes que permitem que cada objeto do teu programa tenha um nome único. Estes criam um sistema onde os nomes num dos teus módulos não entram em conflito com os nomes de outro módulo. Isto torna-os mais úteis.

Estatísticas pypi:

527 487 projetos
5 541 634 versões
10 700 409 ficheiros
802 109 utilizadores
20.3 terabytes de dados

Python 3

```
◎ Uma das linguagens mais utilizadas no mundo
◎ Funciona desde microprocessadores a centros de dados
◎ Fácil de aprender, fácil de ler, fácil de programar
◎ Milhares de bibliotecas disponíveis
◎ Livre e de código aberto
◎ Ferramentas de desenvolvimento livres
```

Projetos em companhias baseadas em Python

```
YouTube
Netflix
Instagram
Spotify
Facebook
PayPal
Amazon
Uber
NASA
Industrial Light and Magic
Entre centenas e centenas de outros...
```

RP2040

O RP2040 inclui um processador Arm Cortex M0 modificado, com dois cores, corre à velocidade de 133MHz, dispõe de 264KB de SRAM e 2MB de armazenamento em memória flash.

Mede 7mm por 7mm com ~3mm de altura.

Pode ser programado MicroPython ou CircuitPython.

Estas são versões otimizadas de Python desenhadas para correr em micro-controladores.

Dispõe de hardware potente capaz de executar até mesmo estruturas de aprendizado de máquina como o TensorFlow Lite. Permite inferência a correr no chip, sem uso de hardware externo.

Outros exemplos incluem vários processadores ARM, CC3200, esp8266, PIC da Microchip, STM32, entre outros.

No mínimo o micro-controlador deve ter:

- 56 KBytes de flash
- 16 Kbytes de RAM
- Relógio de 80 MHz

Estes são requerimentos extremamente conservadores.

Do outro lado da escala podemos encontrar o Facebook, por exemplo.

Dispõe de 85 edifícios em 18 centros de dados, com um total de área aproximadamente 3,7 kilometros quadrados, ou ~687 campos de futebol.

Passamos agora à parte prática deste curso.

Se algo não funcionar ou se tiverem duvidas, é favor interromper e assinalar imediatamente.

É importante que cada passo seja completado com sucesso.

Instalar Python

https://www.python.org/downloads/

V3.10.11

PATH + TK

```
--> python -V
```

```
--> import antigravity
```

```
--> exit()
```

Instalar git

O Git é um sistema de controlo de versões distribuído que rastreia as alterações em qualquer conjunto de ficheiros informáticos, normalmente utilizado para coordenar o trabalho entre programadores.

Os seus objectivos incluem a velocidade, a integridade dos dados e o suporte de fluxos de trabalho distribuídos e não lineares.

https://git-scm.com/downloads

Ativar LFS

--> `git lfs install`

Instalar VSCodium

O Visual Studio Code, também vulgarmente designado como VSCode, é um editor de código-fonte desenvolvido pela Microsoft para Windows, Linux e macOS.

As suas características incluem suporte para depuração, realce de sintaxe, conclusão inteligente de código, refactorização de código e Git incorporado.

Os utilizadores podem alterar o tema, os atalhos de teclado, as preferências e instalar extensões que adicionam funcionalidades.

Neste caso instalamos uma versão mais aberta que remove telemetria e comunicação com a Microsoft.

https://vscodium.com

Instalar Extensões Relevantes

- Python
- Black
- PT_BR
- Pylint

Opção:

- Markdown Editor
- Markdown PDF

Instalar ungoogled chromium

https://github.com/ungoogled-software/ungoogled-chromium

Chromium é uma versão ligeira, livre e de código aberto do browser Chrome desenhada e mantida pela Google.

Sem ter de iniciar sessão numa conta Google, o Chromium funciona muito bem em termos de segurança e privacidade. No entanto, o Chromium ainda tem alguma dependência dos serviços Web e binários da Google.

Para além disso, Google concebeu o Chromium para ser fácil e intuitivo para os utilizadores, o que significa que compromete a transparência e o controlo das operações internas.

O ungoogled chromium modifica o código original das seguintes formas:

- Remove todos os restantes pedidos em segundo plano a quaisquer serviços Web durante a execução do navegador.
- Remove todo o código específico dos serviços Web da Google

- Remove todas as utilizações de binários pré-fabricados do código-fonte e substituí-los por alternativas fornecidas pelo utilizador, sempre que possível.
- Desativa as funcionalidades que inibem o controlo e a transparência e adiciona ou modifica as funcionalidades que as promovam.

Opcional:

-Instalar uBlock Origin e LocalCDN

PIP, C++ Build Tools, Jupyter notebook e pipreqs

--> https://visualstudio.microsoft.com/visual-cpp-build-tools/

PIP é um gestor de pacotes para Python que permite instalar e gerir bibliotecas e dependências que não fazem parte da biblioteca padrão do Python.

--> `pip -V`

Jupyter é um ambiente de desenvolvimento interativo baseado na Web para blocos de notas, código e dados.

Oferece um design flexível e modular, resultados ricos, integração de grandes volumes de dados e suporte para vários utilizadores.

--> `pip install notebook`

--> `jupyter notebook`

--> `pip install pipreqs`

Opcional: Instalar Jupyter lab desktop

https://github.com/jupyterlab/jupyterlab-desktop/releases

Opcional: Instalar em ambiente virtual

--> `python -m venv "venv"`

Trivy

Trivy é o scanner de segurança de código aberto mais popular, confiável, rápido e fácil de usar.
Use o Trivy para encontrar vulnerabilidades e configurações incorretas, riscos de segurança, e muito mais.

https://trivy.dev/

https://github.com/aquasecurity/trivy

Sites relevantes:

```
◎ Github - https://github.com/ ~ CodeBerg - https://codeberg.org/
◎ Pypi - https://pypi.org/
◎ Google Collab - https://colab.research.google.com/
◎ Infinity Free - https://www.infinityfree.com/
◎ Render - https://render.com/
◎ Hugging Face - https://huggingface.co/
◎ Carbon - https://carbon.now.sh/
```

Clonar um repositório:

--> `https://codeberg.org/vluz/MiniCursoPy`

--> `git clone https://codeberg.org/vluz/MiniCursoPy`

Parte prática

...

Recursos Adicionais

Think Python:

https://www.greenteapress.com/thinkpython/thinkpython.pdf

Cursos livres no YouTube:

https://www.youtube.com/watch?v=_uQrJ0TkZlc

https://www.youtube.com/watch?v=eWRfhZUzrAc

IOS e Android

https://apps.apple.com/us/app/carnets-jupyter/id1450994949

https://apps.apple.com/us/app/libterm/id1380911705

https://termux.dev/en/

--> `pkg install python`

https://dev.to/codeledger/how-to-get-visual-studio-code-to-run-in-termux-on-android-405j

---

OBRIGADO!

Dedicado à Lana por suporte e paciência.