Algoritmos e Programação

Disciplina oferecida para os cursos:

1º ano de Engenharia

4 créditos - 72 horas - Disciplina de 36 encontros

2º Semestre de 2025

FACAMP

Sobre o professor

Thiago José Cóser é Mestre em Artes Visuais pela Universidade Estadual de Campinas, tem como foco de trabalho o encontro entre Design e Tecnologia. Professor, Designer e desenvolvedor de diversos projetos que envolvem novas mídias como Jogos Digitais, Realidade Aumentada, Realidade Virtual, Modelagem 3D, Desenvolvimento de aplicativos, ludificação, entre outros. Possui conhecimento pleno de diversos softwares para produções multimídia.

Contato
thiago.coser@facamp.com.br

Linkedin
https://www.linkedin.com/in/thiagocoser

Repositório Github do curso
https://github.com/ThiagoCoser/cursoPython

Observações gerais

- Calendário acadêmico: observar datas importantes, como provas!
- Chamada: avisar caso tenha algum erro, não matriculado, RA ou nome errado, etc., AVISAR!

Repositório das aulas

Grande parte do material do curso, com explicações, exemplos, exercícios e scripts em Python podem ser encontrados no repositório abaixo:

https://github.com/ThiagoCoser/cursoPython2025

Ementa

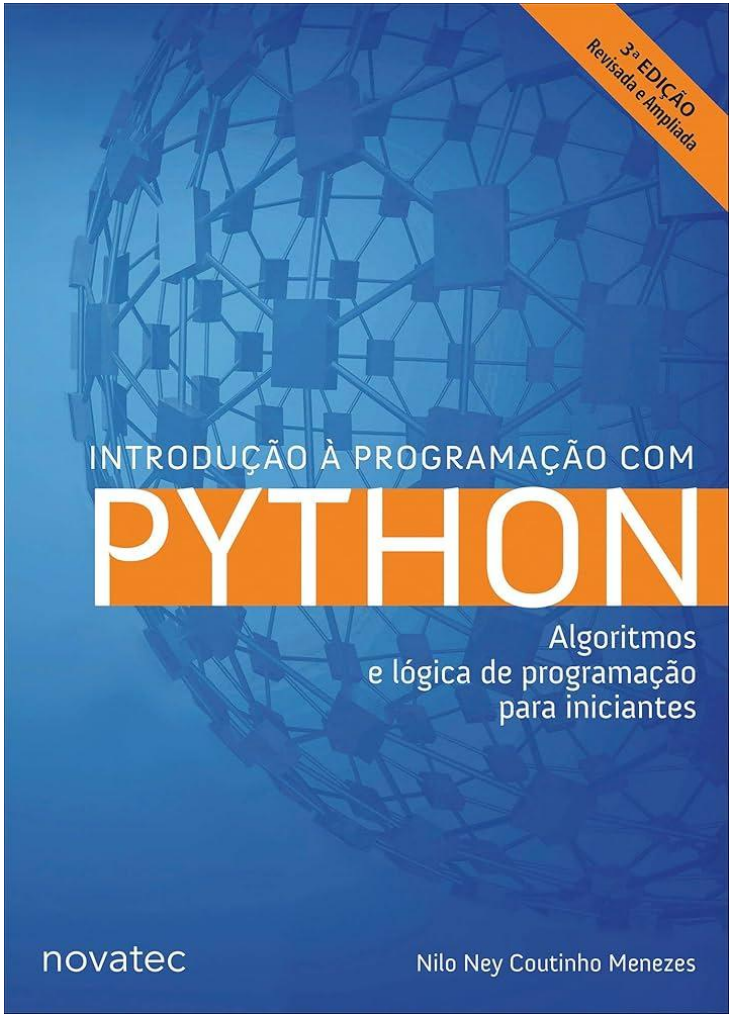
Algoritmos e Programação

Conceitos básicos sobre computadores. Variáveis e tipos básicos de dados. Operações primitivas. Expressões aritméticas e lógicas. Conceito de algoritmo. Linguagem narrativa, fluxograma e linguagem algorítmica. Estruturas básicas de programas: sequencial, condicional e de repetição. Tipos de dados compostos homogêneos: vetores e matrizes.

Bibliografia básica

DEITEL, P., DEITEL, H. Java TM: como programar. 8a ed. São Paulo: Pearson Prentice Hall, 2012.
FORBELLONE, André Luiz Villar. Lógica de programação. São Paulo: Pearson Prentice Hall, 2005. 218 p.
MANZANO, José Augusto; N. G.; OLIVEIRA, Jayr Figueiredo. Estudo dirigido de algoritmos. São Paulo: Érica, 2011.
COMPLEMENTAR
DASGUPTA, Sanjoy PAPADIMITRIOU, Christos;VAZIRANI,Umesh. Algoritmos. Porto Alegre: AMGH, 2010.
EDELWEISS, Nina; LIVI, Maria Aparecida Castro. Algoritmos e programação com exemplos em Pascal e C. Porto Alegre: Bookman, 2014.
GOODRICH, Michael T.; TAMASSIA Roberto. Estruturas de dados e algoritmos em Java. Porto Alegre: Bookman, 2013.
WIRTH, Niklaus. Algoritmos e estruturas de dados. Rio de Janeiro: LTC, 2012.
ZIVIANI, Nivio. Projeto de algoritmos. São Paulo: Cengage Learning, 2013.

Bibliografia básica



Metodologia

Aulas expositivas e exercícios práticos em sala.

Notas e avaliações

A nota do curso será baseada:

- primeira prova semestral (final agosto), valendo (30% da nota total)
- segunda prova semestral (final de setembro), valendo (30% da nota total)
- entrega de um trabalho individual ou em duplas (final outubro) (30% da nota total)
- entrega de listas de exercícios (10% da nota total)

Metodologia

Sobre o trabalho

Individualmente ou em duplas, o trabalho tem tema livre, e deverá ser apresentado na semana da segunda prova bimestral. A nota deste trabalho será baseada na originalidade, documentação e qualidade de execução e apresentação.

Motivação

Porque aprender lógica de programação?

A lógica de programação é a habilidade de criar sequências lógicas de instruções que um computador pode seguir para resolver um problema. Também auxilia resolução de problemas de maneira mais abrangente, o que contribui para o pensamento crítico e operacionalizar melhor problemas de maneira gerenciável.

Motivação

Está tudo bem em errar!

Os programas irão falhar se contiverem código que o computador não consegue entender, o que fará com que o Python mostre uma mensagem de erro e seu algoritmo interrompido. Então, não tenha medo de cometer erros: uma falha significa apenas o programa parou de funcionar inesperadamente e que seu computador não vai explodir. O console, terminal, ou IDE vão mostrar qual o erro você cometeu, tente pesquisar sobre a linha de erro.

Motivação

Na real, os programadores não precisam saber muita matemática.

A ansiedade mais comum ao iniciar os estudos em programação é as pessoas acreditam que precisam ser muito boas em matemática. Na verdade, a maior parte da programação não requer matemática além da aritmética básica. Programar está mais para ser bom em resolver quebra-cabeças ou cozinhar uma boa refeição :)

Motivação

Então, você quer aprender a programar?

Responda com calma a estas perguntas:

1. Você quer aprender a programar?
2. Como está seu nível de paciência?
3. Quanto tempo você pretende estudar?
4. Qual o seu objetivo ao programar?

Porque Python?

- Uso geral
- Código aberto sob a Licença Python, permissiva
- Fácil legibilidade
- Suporta diferentes paradigmas de programação como estruturada (procedural) ou orientada a objeto
- Baterias inclusas (ampla biblioteca padrão)
- Criação de gráficos e visualizações de maneira fácil
- Forte comunidade, diversas bibliotecas
- Criação de Jogos (Pygame, Godot)
- Aplicativos (Desktop e Mobile)
- Machine Learning, IA
- Data science
- Web, criação de APIs (django, Flask, FastAPI)
- Automatização de tarefas (renomear pastas, automatizar e mails, testes, etc)
- Creative code

Introdução

O que é um algoritmo?

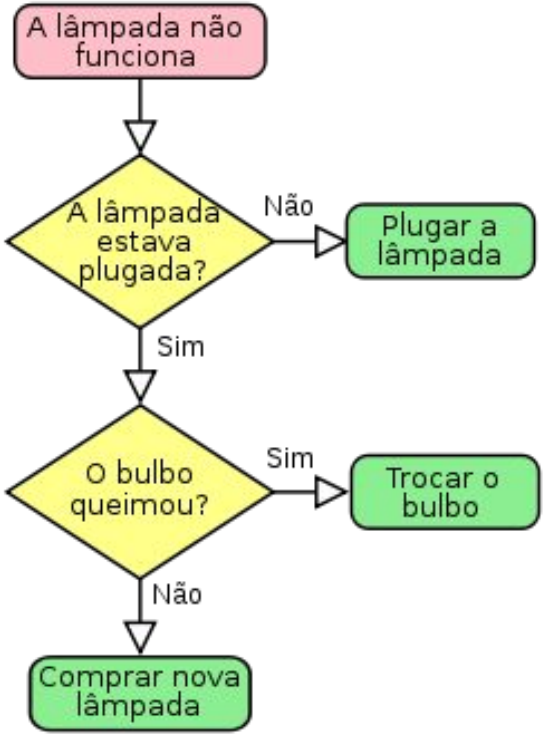
Introdução

Vamos trocar uma lâmpada?

Faça o algoritmo de como trocar uma lâmpada.

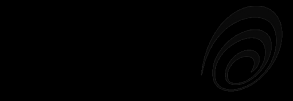
Introdução

Vamos trocar uma lâmpada?



Em Py

```python
# Algoritmo para trocar uma lâmpada

# Função para verificar o estado da lâmpada
def trocar_lampada(esta_queimada):

    # Verifica se a lâmpada está queimada
    if esta_queimada:
        print("A lâmpada está queimada.")
        print("Substituindo por uma nova lâmpada...")
        # Aqui poderia ter o código para substituir a lâmpada
        print("A nova lâmpada foi instalada.")
    else:
        print("A lâmpada está funcionando. Não é necessário trocá-la.")

# Exemplo de uso
esta_queimada = True  # True indica que a lâmpada está queimada
trocar_lampada(esta_queimada)
```



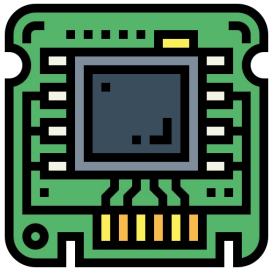
Crossfit deveria ser proibido

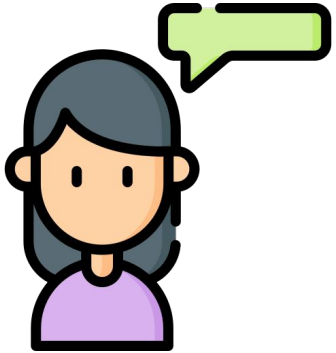
Vá devagar, o caminho é legal

```
# Isto irá mostrar uma mensagem
print ("Olá mundo!")
```

Sobre o uso do ChatGPT

Interpretador





Instalação

- MAC e Lunix já tem instalado
- Windows - https://www.python.org

Documentação
https://docs.python.org/3/
https://wiki.python.org/moin/BeginnersGuide

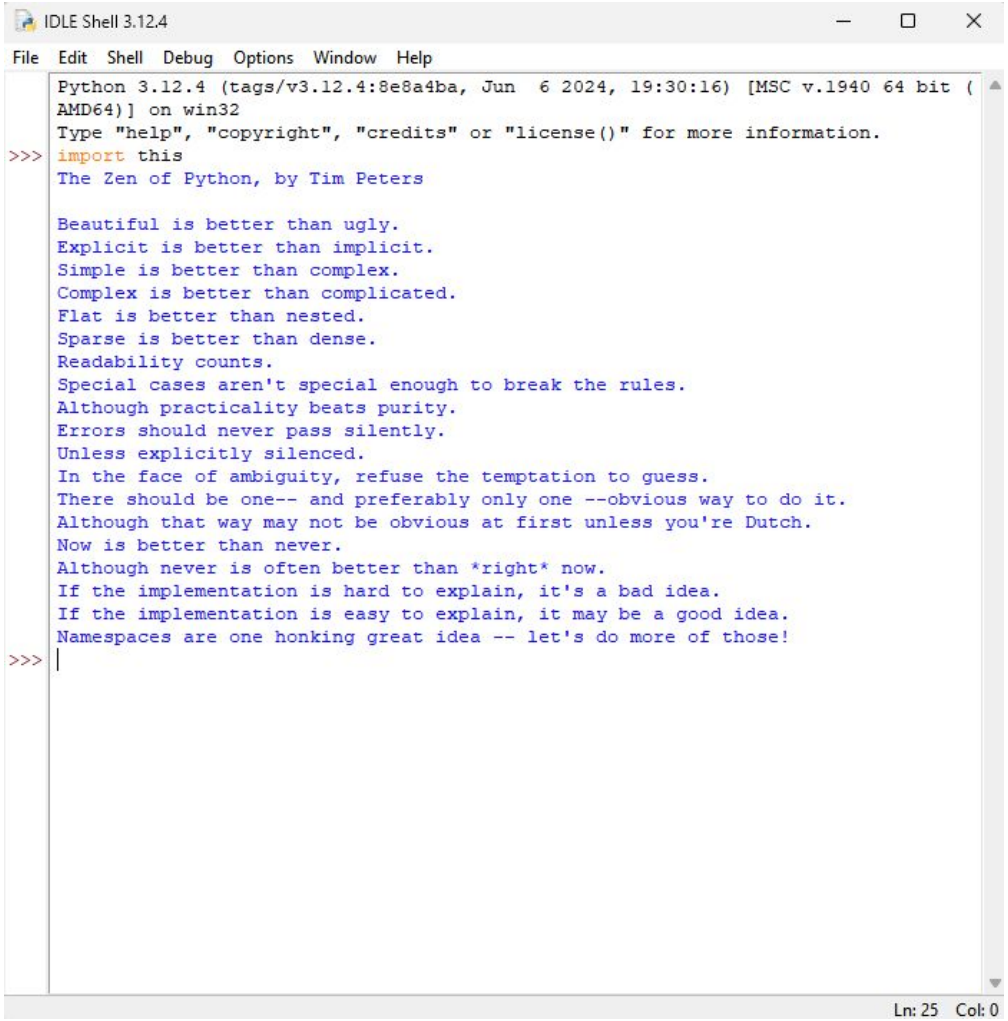
Build in functions
https://docs.python.org/3/library/functions.html

Primeiro uso

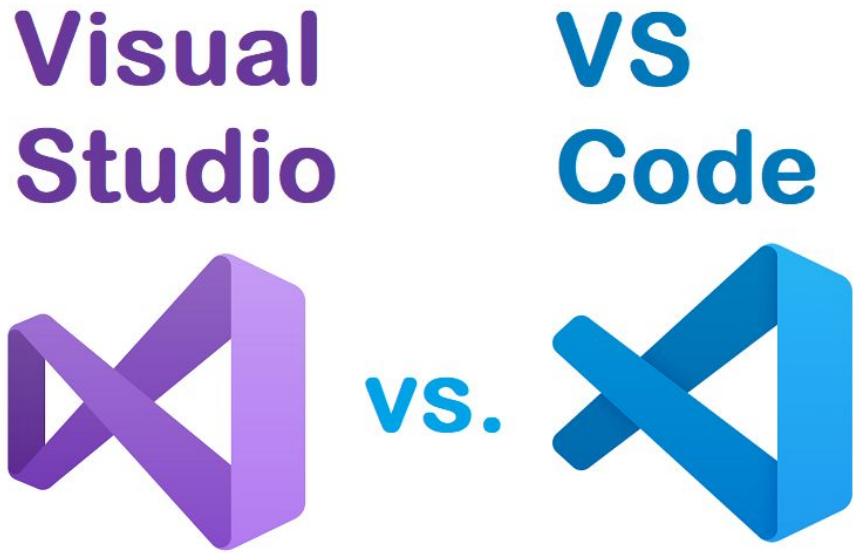
Windows
via prompt (cmd) >Python REPL (Read-Eval-Print Loop), IDLE ou IDE (Vs Code ou Pycharm)

Comandos iniciais para verificar instalação

- python
- python --version
- import this


IDLE Shell 3.12.4
File Edit Shell Debug Options Window Help
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun 6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
Ln: 25 Col: 0

IDE (Ambiente de Desenvolvimento Integrado




PyCharm
O IDE Python para ciência de dados e desenvolvimento Web
Torne o desenvolvimento mais produtivo e agradável
Baixar
Edição Professional completa ou Community gratuita
PC

Extensões Vs Studio Code

- Python
- Pylance
- IntelliCode
- Jupyter
- Dracula

Extension ID

- ms-python.python
- ms-python.vscode-pylance
- VisualStudioExptTeam.vscodeintellicode
- ms-toolsai.jupyter-keymap
- dracula-theme.theme-dracula

Bibliotecas

Mesmo Python vindo com as baterias inclusas, há inúmeras bibliotecas disponíveis, como veremos mais tarde no curso.

The Python Standard Library
https://docs.python.org/3/library/index.html

Install Documentation Learn Community About Us News Contribute English

NumPy

The fundamental package for scientific computing with Python

LATEST RELEASE: NUMPY 1.26. VIEW ALL RELEASES

NumPy 2.0 release date: June 16 2024-05-23

Powerful N-dimensional arrays

Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

Numerical computing tools

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

Open source

Distributed under a liberal BSD license, NumPy is developed and maintained publicly on GitHub by a vibrant, responsive, and diverse community.

Interoperable

NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

Performant

The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

Easy to use

NumPy's high level syntax makes it accessible and productive for programmers from any background or experience level.

https://numpy.org/

Plotly Open Source Graphing Library for Python

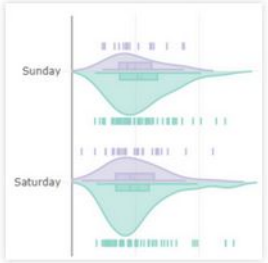
Plotly's Python graphing library makes interactive, publication-quality graphs. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, polar charts, and bubble charts.

Plotly.py is free and open source and you can view the source, report issues or contribute on GitHub.

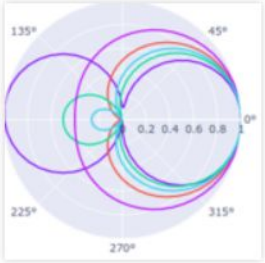
> Deploy Python AI Dash apps on private Kubernetes clusters: Pricing | Demo | Overview | AI App Services

Fundamentals

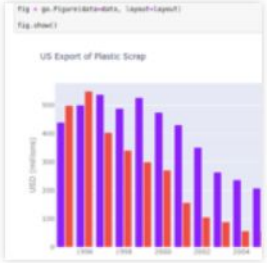
More Fundamentals »



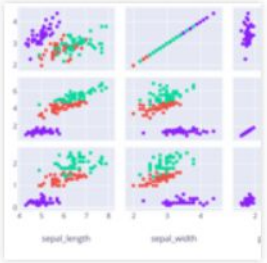
The Figure Data Structure



Creating and Updating Figures



Displaying Figures



Plotly Express

Analytical Apps with Dash

https://plotly.com/python/

https://www.pygame.org/

Principais operadores

Operadores possibilitam transcrever a lógica para código. Experimente iniciar utilizando o interpretador como calculadora.

```
>>> 2+3
5

>>> 3-1
2
```

Operadores Aritméticos

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
//	Divisão inteira
%	Módulo
**	Exponenciação

Operadores de Comparação

==	Igual a
!=	Diferente de
>	Maior que
>=	Maior ou igual
<	Menor que
<=	Menor ou igual

Operadores de Atribuição

`=`	Atribuição: `a = b` - Atribui o valor de `b` à variável `a`
`+=`	Atribuição com soma: `a += b` - Equivalente a `a = a + b`
`-=`	Atribuição com subtração: `a -= b` - Equivalente a `a = a - b`
`*=`	Atribuição com multiplicação: `a *= b` - Equivalente a `a = a * b`
`/=`	Atribuição com divisão: `a /= b` - Equivalente a `a = a / b`.
`//=`	Atribuição com divisão inteira: `a //= b` - Equivalente a `a = a // b`
`%=`	Atribuição com módulo: `a %= b` - Equivalente a `a = a % b`
`**=`	Atribuição com exponenciação: `a **= b` - Equivalente a `a = a ** b`

Operadores Lógicos

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

Operadores de Identidade

`is`	(É): `a is b` - Verifica se `a` e `b` referem-se ao mesmo objeto.
`is not`	(Não é) : `a is not b` - Verifica se `a` e `b` não referem-se ao mesmo objeto.

Operadores de Associação

in (Em): a in b - Verifica se a está contido em b (como em uma lista ou string).
not in (Não em): a not in b - Verifica se a não está contido em b.

Variáveis

Variáveis permitem armazenar diversos tipos de dados. Cada tipo de variável tem suas próprias características e métodos associados, permitindo diferentes operações e manipulações de dados. Aqui estão os principais tipos de variáveis e estruturas de dados em Python:

Números Inteiros (int)
x = 10

Números de Ponto Flutuante (float)
y = 3.14

Strings (str)
nome = "João"

Booleanos (bool)
ativo = True

Variáveis

Listas (list)
frutas = ["maçã", "banana", "laranja"]

Tuplas
ponto = (10, 20)

Conjuntos
numeros = {1, 2, 3, 4, 5}

Dicionários
pessoa = {"nome": "Alice", "idade": 30}

Podemos ter diversos outros tipos como imagens ou sons como variáveis

Variáveis

Listas (list)
frutas = ["maçã", "banana", "laranja"]

Tuplas
ponto = (10, 20)

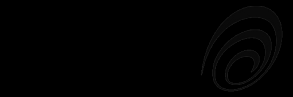
Conjuntos
numeros = {1, 2, 3, 4, 5}

Dicionários
pessoa = {"nome": "Alice", "idade": 30}

Podemos ter diversos outros tipos como imagens ou sons como variáveis

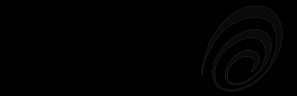
Variáveis

```
# Lista de notas dos alunos
notas = [7.5, 8.0, 6.5, 9.0, 7.0]

# Calculando a média das notas
media_notas = sum(notas) / len(notas)

# Verificando se a média é maior que 7 para aprovação
if media_notas > 7:

    print("Aluno aprovado!")
else:
    print("Aluno reprovado!")
```



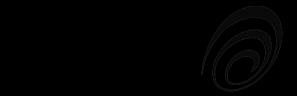
Descubra o erro

```
numeroA = input ("Digite um número A")
numeroB = input ("Digite um número B")

print (numeroA+numeroB)
```



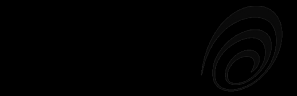
Descubra o erro

```
salario = 1000
aumento = 15%
print (salario+aumento)
```



Descubra o erro

```
import time

tempo = 10

while (tempo>0):
    tempo-=1
    print(tempo)

time.sleep(1)
    print ("Terminou")
```



Descubra o erro

```
nome = ""

while (nome != "seu nome"):
    print("Por favor, digite seu nome")
    nome = input()

print("Obrigado!")
```

Controle de Fluxo I

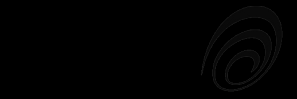
Um dos super poderes de um código não é apenas executar uma instrução após a outra, mas com base em como as expressões são avaliadas, o programa pode tomar ações como pular instruções, repeti-las, ou executar outras ações.

Instruções de controle de fluxo definem as condições para as tomadas de ações dentro do seu script em Python, com sintaxes como “se” (If), “senão se” (Elif) ou “senão” (Else), definem como o script deve continuar em casos de terminadas condições serem atingidas.

Ver item **04.ControleDeFluxo_I.py** no repositório.

Controle de Fluxo I

```
# Exemplo I
# Este algoritmo ordena dois números

a = 5
b = 7
if a > b:
    print("a é maior que b")
elif a < b:
    print("a é menor que b")
else:
    print("a e b são iguais")
```



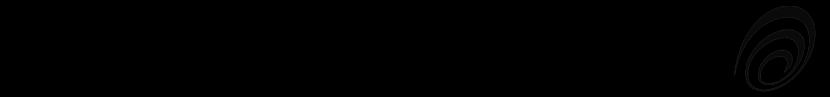
Controle de Fluxo II

Outro super poder é criar repetições, estruturas em *looping* que serão executadas até uma determinada condição ser atingida. Exemplos incluem a sintaxe “Enquanto” (While) e “Para” (For). Com isto podemos realizar tarefas em grandes escalas

Ver item **06.ControleDeFluxo_II.py** no repositório.

Controle de Fluxo I

```
# Este algoritmo cria uma contagem regressiva
import time

tempo = 1

while (tempo>0):
    tempo-=1
    print(tempo)
    time.sleep(1)

print ("Terminou")
```



Exercícios I (organize cada um em um .py separado)

1) Imprima “Olá mundo!”
2) Faça um programa que tenha as variáveis de dia, mês e ano e imprima, formatando a data com o nome da cidade e “/” entre as variáveis.
3) Faça um programa com duas variáveis de números inteiros, atribua valores a elas e imprima o resultado da soma, subtração, multiplicação e divisão entre elas.
4) Faça um programa que peça o nome, idade e altura de uma pessoa e imprima os valores, formatados, com uma mensagem personalizada
5) Faça um programa que peça um valor em metros e o mostre-o convertido em milímetros
6) Faça um programa que peça um valor de temperatura em Celsius e mostre o valor convertido para Fahrenheit. (cálculo para Fahrenheit: F = 9*C/5+32)
7) Faça um programa que calcule o aumento de um salário. Ele deve solicitar o valor do salário e a porcentagem de aumento. Mostre o valor do aumento, do salário atual e quanto a mais este aumento vai resultar em 1 ano.
8) Peça 4 notas de um aluno. Tire a média e mostre se ele foi aprovado ou não (média 7).
9) Faça um programa com duas variáveis de números inteiros, atribua valores de 5 e 3 a elas. Imprima o resultado da divisão aproximada delas. Dica: utilize a função round()
10) Imprima (“Terminei, estou adorando programar!”)

Exercícios II (organize cada um em um .py separado)

1) Faça um dado D6 e imprima o valor aleatório sorteado. Dica: utilize o módulo random e a função randint()
2) Crie uma lista com os números de 1 a 100. A seguir, escolha um número desta lista e imprima-o. Utilize a função choice() do módulo random
3) Crie um programa que solicite um número ao usuário e informe se o número é par ou ímpar. Dica: Use o operador % para verificar a divisão inteira
4) Crie um programa que exiba uma contagem regressiva de 100 para 0. Dica: Use um loop while
5) Calcule a soma dos números de 1 a 100 e exiba o resultado.
6) Peça ao usuário para inserir um número e exiba a tabuada desse número de 1 a 10. Dica: Use um loop for em um range e a operação de multiplicação.
7) Use um loop for e range e imprima todos os números pares de 1 a 100
8) Use um loop for e range para somar todos os números ímpares de 1 a 300 e exiba a soma
9) Crie uma lista com 3 nomes, peça um nome para o usuário e diga se este nome está contido ou não na lista.

Extras:

10) Crie um programa que gere e exiba os primeiros 10 números da sequência de Fibonacci. Dica: use uma estrutura de loop para gerar a sequência.
11) Melhore o exercício 1 solicitando um número ao usuário. Adicione uma checagem de erro caso ele não digite um número
12) Melhore o exercício 4 com o módulo Time e a função sleep()

Exercícios III (organize cada um em um .py separado)

1) Faça dois scripts (A e B). Crie uma função no scriptA, importe-o como um módulo no scriptB e ative a função de A, a partir de valores no scriptB;
2) Crie uma mensagem de boas vindas e formate-a em maiúsculo;
3) Crie uma string com espaços extras e remova-os. Exemplo: texto = " Espaços extras "
4) Utilize a função replace() e troque azul por nublado da variável texto = "O céu está azul". Formate com a função dentro de uma f-string.
5) Dado a variável meuTexto = “Esta lista contém muitos nomes”, utilize uma f-strig com if e else para dizer se a variável meuTexto contém a string “muitos”
6) Dada a variável texto = "Python é incrível", utilize uma f-string para cortar o texto e imprimir somente a primeira palavra da frase
7) Crie uma variável com 10 dígitos de π (Pi) e formate os apenas os 2 primeiros dígitos, com vírgula, utilizando um f-string.
8) Conte o número de vezes que a letra r aparece na frase: “O rato roeu a roupa do rei de Roma”. Conte diretamente em um f-string

Importando módulos

Todos os programas Python podem chamar um conjunto básico de funções chamadas funções integradas, incluindo as funções print(), input() e len() que você viu antes. Python também vem com um conjunto de módulos denominado biblioteca padrão. Cada módulo é um programa Python que contém um grupo relacionado de funções que podem ser incorporado em seus programas. Por exemplo, o módulo matemático (math) possui um conjunto de funções matemáticas, o módulo aleatório (random) tem funções relacionadas a números aleatórios, o módulo (time) possui funções de tempo, o módulo de teclado (keyboard) permite utilizar as teclas do teclado e assim por diante.

Importando módulos

Antes de poder usar as funções em um módulo, você deve importar o arquivo módulo com uma instrução de importação. No código, uma instrução de importação consiste em:

• A palavra-chave de importação (import)

• O nome do módulo (exemplo, import time)

• Opcionalmente, mais nomes de módulos, desde que separados por vírgulas

Depois de importar um módulo, você pode usar todas as funções interessantes desse módulo.

módulo. Vamos tentar com o módulo aleatório, que nos dará acesso a função random.ranint()

Usando módulos

```
# Sorteia um número aleatório utilizando a função random.ranint() do módulo random

import random

numeroAleatorio = random.randint(1, 10)
print(numeroAleatorio)
```