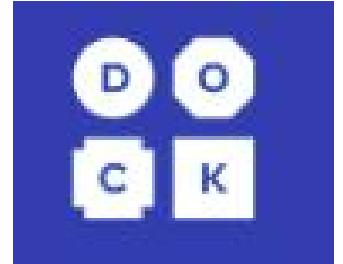


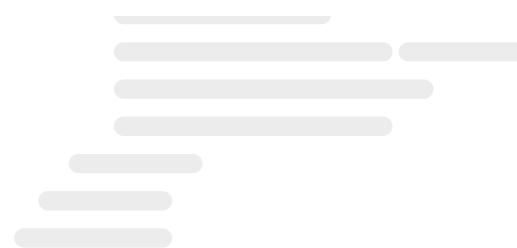


Universidade  
**PYTHON**



**1<sup>a</sup> Turma - ADA LOVELACE**

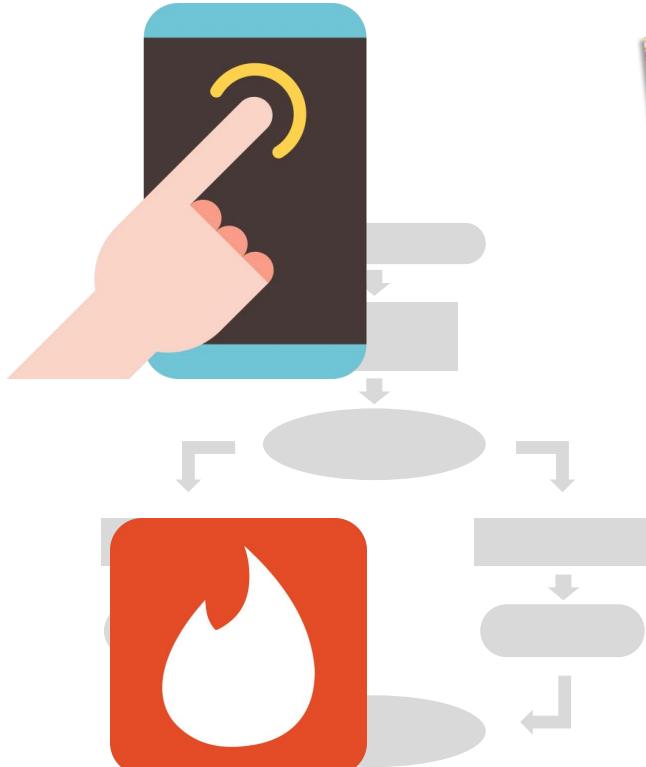
Bem vindo!!





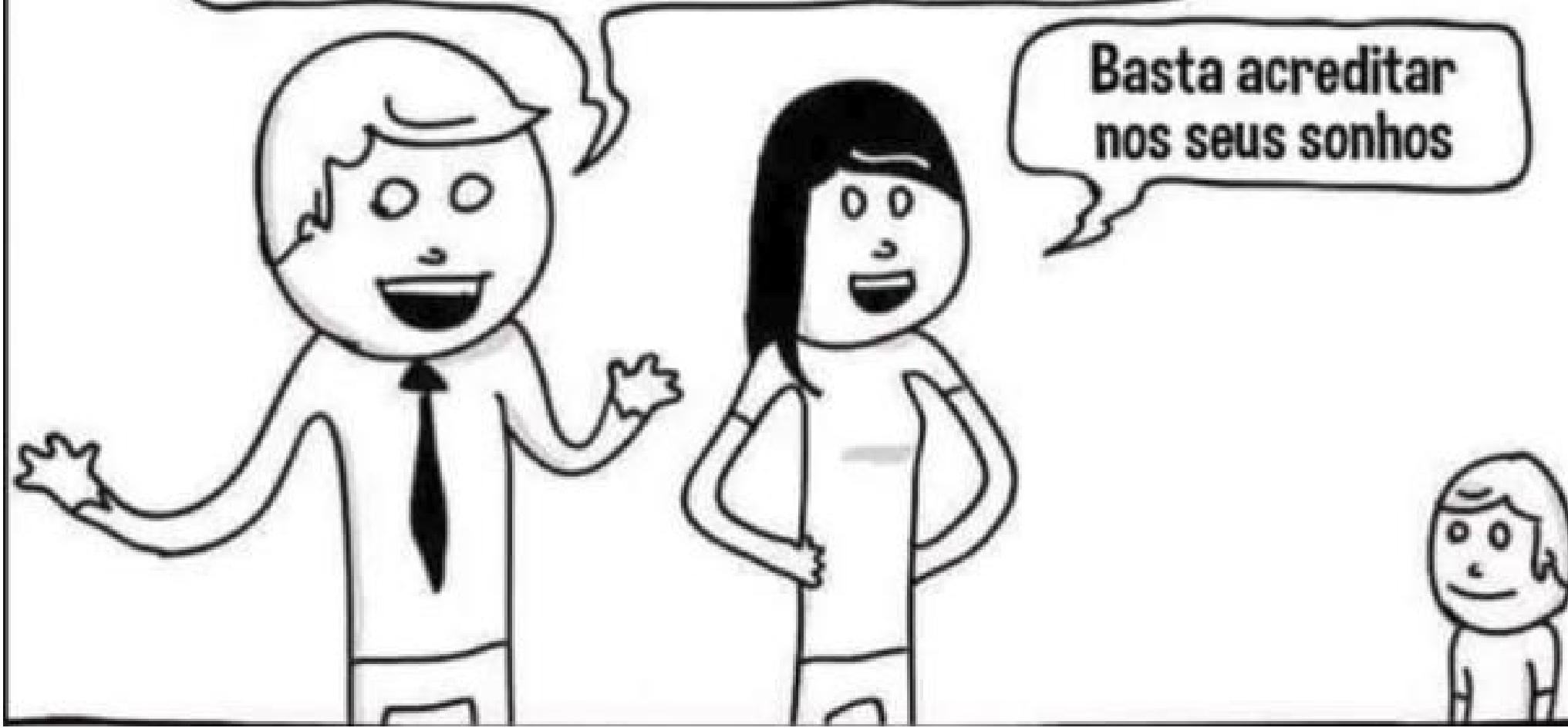
Parabéns!

# O que aconteceu com o mundo?



**Você é capaz de fazer o que quiser  
e vai conquistar o mundo!**

**Basta acreditar  
nos seus sonhos**



# 10 ANOS DEPOIS



TRANSFORMADOR  
DE LIMÃO EM  
LIMONADA



# Outliers



THE STORY OF SUCCESS

MALCOLM  
GLADWELL

#1 bestselling author of *The Tipping Point* and *Blink*

**10.000**

**HORAS**

I ❤ 1991



# Porquê Python?

Seu nome é uma homenagem ao grupo humorístico inglês Monty Python, adorado por geeks e hackers de todo o mundo.



Foi lançada por [Guido van Rossum](#) em [1991](#)





```
from mininet.topo import Topo

class Router_Topo(Topo):
    def __init__(self):
        "Create P2P topology."

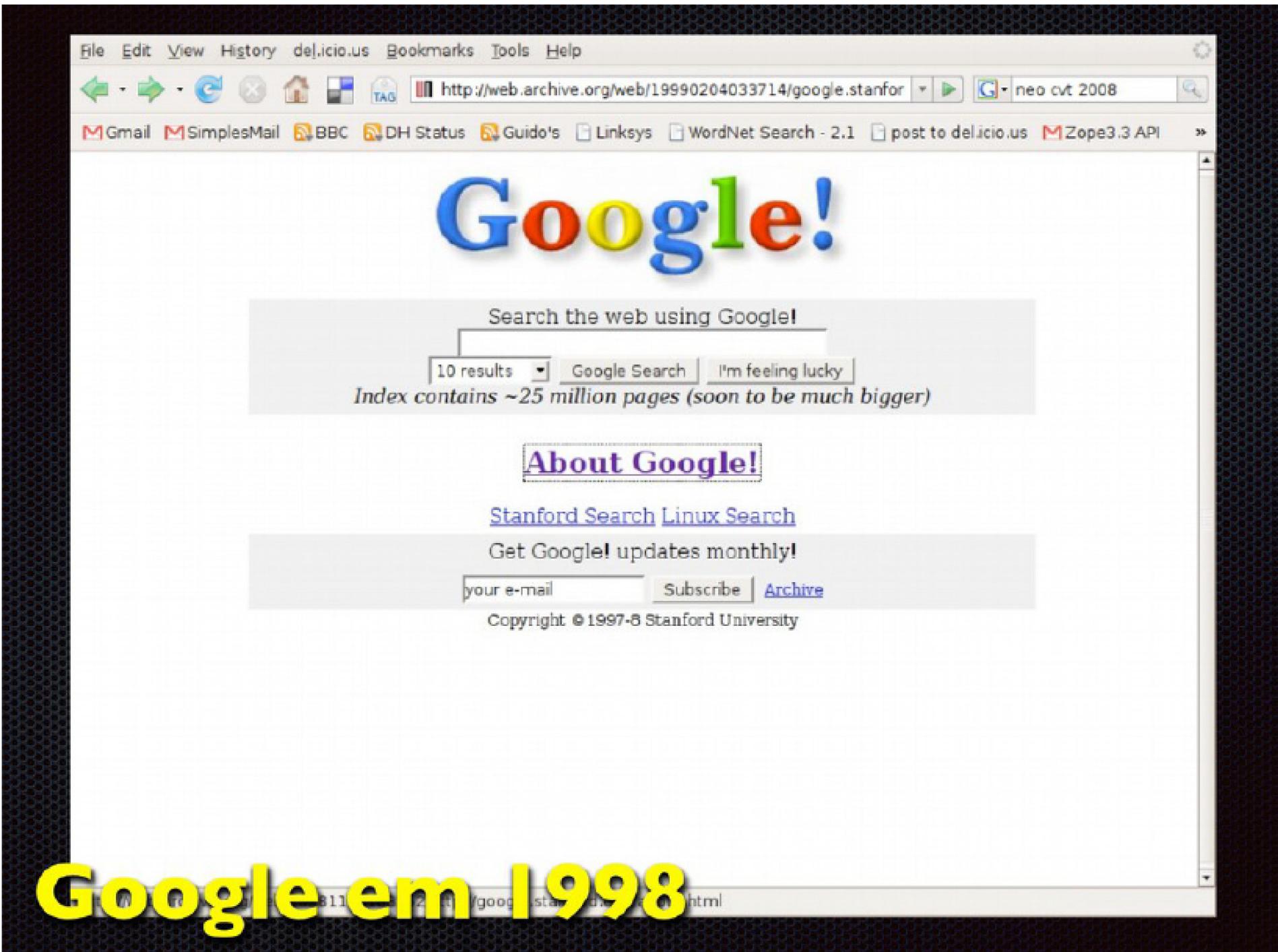
        # Initialize topology
        Topo.__init__(self)

        # Add hosts and switches
        H1 = self.addHost('h1')
        H2 = self.addHost('h2')
        H3 = self.addHost('h3')
        S1 = self.addSwitch('s1')
        S2 = self.addSwitch('s2')

        # Add Links
        self.addLink(H1, S1)
        self.addLink(H2, S1)
        self.addLink(H2, S2)
        self.addLink(H3, S2)

topos = {
    'router': (lambda: Router_Topo())
}
```





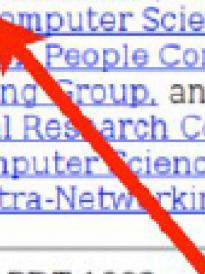
# About Google!

## Links

- [Research Papers about Google and the WebBase](#)
- [Pictures and stats for the Stanford Google Hardware](#)

## Credits

- Current Development: [Sergey Brin](#), [Larry Page](#), and [Craig Silverstein](#)
- Design and Implementation Assistance: [Scott Hassan](#) and [Alan Steremberg](#)
- Faculty Guidance: [Hector Garcia-Molina](#), [Rajeev Motwani](#), [Jeffrey D. Ullman](#), and [Terry Winograd](#)
- Research Funding: [NSE](#), [NASA](#), [DARPA](#) and [Interval Research](#)
- Equipment Donations: [IBM](#), [Intel](#), and [Sun](#)
- Equipment Consulting: [Penguin Computing](#)
- Software: [GNU](#), [Linux](#), [Python](#), [Parasoft](#) (debugging), and [Gimp](#) (logo design)
- Collaborating Groups in the [Computer Science Department](#) at [Stanford University](#): [The Digital Libraries Project](#), [The Project X People](#), [Computers and Design](#), [The Database Group](#), [The Stanford InfoLab](#), [The MIDAS Data Mining Group](#), and [The Theory Division](#)
- Outside Collaborators: [Interval Research Corporation](#) and the [IBM Almaden Research Center](#)
- Technical Assistance: [The Computer Science Department's Computer Facilities Group](#), [Stanford's Distributed Computing](#) and [Intra-Networking Systems Group](#)





INDUSTRIAL  
LIGHT & MAGIC

ABOUT ILM

OUR WORK



# Porquê Python?

Faça ou não faça, tentativa não há...



**genba** **benja** **sen**

# COBOL

# Perl Python SmallTalk

C++

# Java



```
import java.io.*;
import java.util.*;

public class LW implements Comparator{
    public static void main(String[] args) {
        LW lw = new LW();
        lw.somaExtensoes(args[0]);
    }

    public void somaExtensoes(String arq) {
        try {
            BufferedReader reader = new BufferedReader(new FileReader(arq));
            String linha;
            HashMap dict = new HashMap();
            while ((linha = reader.readLine()) != null) {
                String ext = linha.substring(linha.lastIndexOf('.')+1,linha.length());
                Integer i = (Integer)dict.get(ext);
                if (i == null) i = new Integer(0);
                dict.put(ext,new Integer(i.intValue() + 1));
            }

            TreeSet ts = new TreeSet(this);
            ts.addAll(dict.entrySet());
            Iterator it = ts.iterator();
            while (it.hasNext()) {
                Map.Entry me = (Map.Entry)it.next();
                System.out.println(me.getKey() + " - " + me.getValue());
            }
        } catch (Exception e) {
            System.out.println("Exception e: " + e);
            e.printStackTrace();
        }
    }

    public int compare (Object o1, Object o2) {
        int i1 = ((Integer)((Map.Entry)o1).getValue()).intValue();
        int i2 = ((Integer)((Map.Entry)o2).getValue()).intValue();
        return i2 - i1;
    }
}
```

# Java



```
import java.io.*;
import java.util.*;

public class LW implements Comparator{
    public static void main(String[] args) {
        LW lw = new LW();
        lw.somaExtensoes(args[0]);
    }

    public void somaExtensoes(String arq) {
        try {
            BufferedReader reader = new BufferedReader(new FileReader(arq));
            String linha;
            HashMap dict = new HashMap();
            while ((linha = reader.readLine()) != null) {
                String ext = linha.substring(linha.lastIndexOf('.')+1,linha.length());
                Integer i = (Integer)dict.get(ext);
                if (i == null) i = new Integer(0);
                dict.put(ext,new Integer(i.intValue() + 1));
            }

            TreeSet ts = new TreeSet(this);
            ts.addAll(dict.entrySet());
            Iterator it = ts.iterator();
            while (it.hasNext()) {
                Map.Entry me = (Map.Entry)it.next();
                System.out.println(me.getKey() + " - " + me.getValue());
            }
        } catch (Exception e) {
            System.out.println("Exception e: " + e);
            e.printStackTrace();
        }
    }

    public int compare (Object o1, Object o2) {
        int i1 = ((Integer)((Map.Entry)o1).getValue()).intValue();
        int i2 = ((Integer)((Map.Entry)o2).getValue()).intValue();
        return i2 - i1;
    }
}
```

# Java

# Python

```
#!/usr/bin/env python
linhas = open('massa.txt').readlines()
dic_ext = {}
for linha in linhas:
    ext = linha.split('.')[1]
    ext = ext.strip()
    dic_ext[ext] = dic_ext.get(ext,0) + 1

rel = dic_ext.items()
rel = [ (qtd, ext) for (ext, qtd) in rel ]
rel.sort()
rel.reverse()
for item in rel:
    print '%3d %s' % item
```

# O que é Python?

- Prototipagem rápida de linguagem OO
- Não é apenas uma linguagem de scripting
- Não apenas um outro Perl
- Extensível
- Possível incorporar em aplicações
- Extremamente portátil.
- Compilado para código interpretador
- Compilação é implícita e automática
- Gerenciamento automático de memória
- ...



# Porquê Python?

```
and      del      from      not      while
as       elif     global    or       with
assert   else     if        pass    yield
break   except   import   print
class   exec    in        raise
continue finally  is        return
def     for     lambda
```

# 31 Palavras



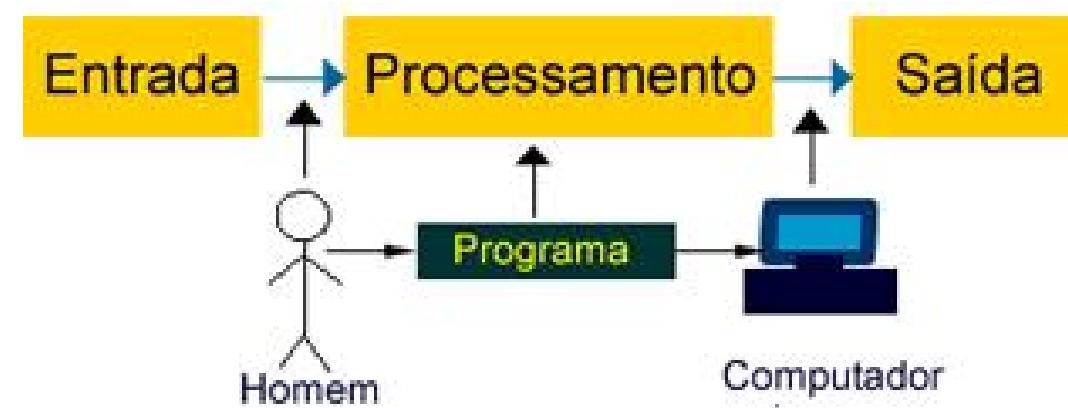
# Python no mundo



A close-up photograph of a young baby with light brown hair, looking directly at the camera with a neutral expression. The baby is wearing a white t-shirt with a green collar and a green sleeve visible on the left. A small, pink, textured object, possibly a piece of coral or a shell, is held in the baby's right hand near the bottom left corner of the frame. The background is a blurred, sandy beach with some water and sky visible.

# Aprenda PYTHON

# TUDO QUE VOCÊ PRECISA SABER



# Lógica de Programação

## LÓGICA DE PROGRAMAÇÃO

**LÓGICA É A MANEIRA COM A QUAL RESOLVEMOS UM DETERMINADO PROBLEMA**

## ALGORITMOS

**É UM CONJUNTO DE INSTRUÇÕES ORGANIZADAS LOGICAMENTE PARA A RESOLUÇÃO DE DETERMINADOS PROBLEMAS**

**TUDO COMEÇA QUANDO VOCÊ APRENDER  
ALGORITMO E LÓGICA DE PROGRAMAÇÃO.**



# POR QUE NÃO APRENDO?

$$2x^2 + 1x + 4 = 0$$

$$x = \frac{-b \pm \sqrt{\Delta}}{2.a}$$

INTERPRETAR <> ENTENDER

# Lógica de Programação

## AÇÃO

INICIO

EXIBIR

LER

PROCESSAR

VERIFICAR

CRIAR

SE

REPETIR

ENQUANTO

ARMAZENAR

CONSULTAR

FIM

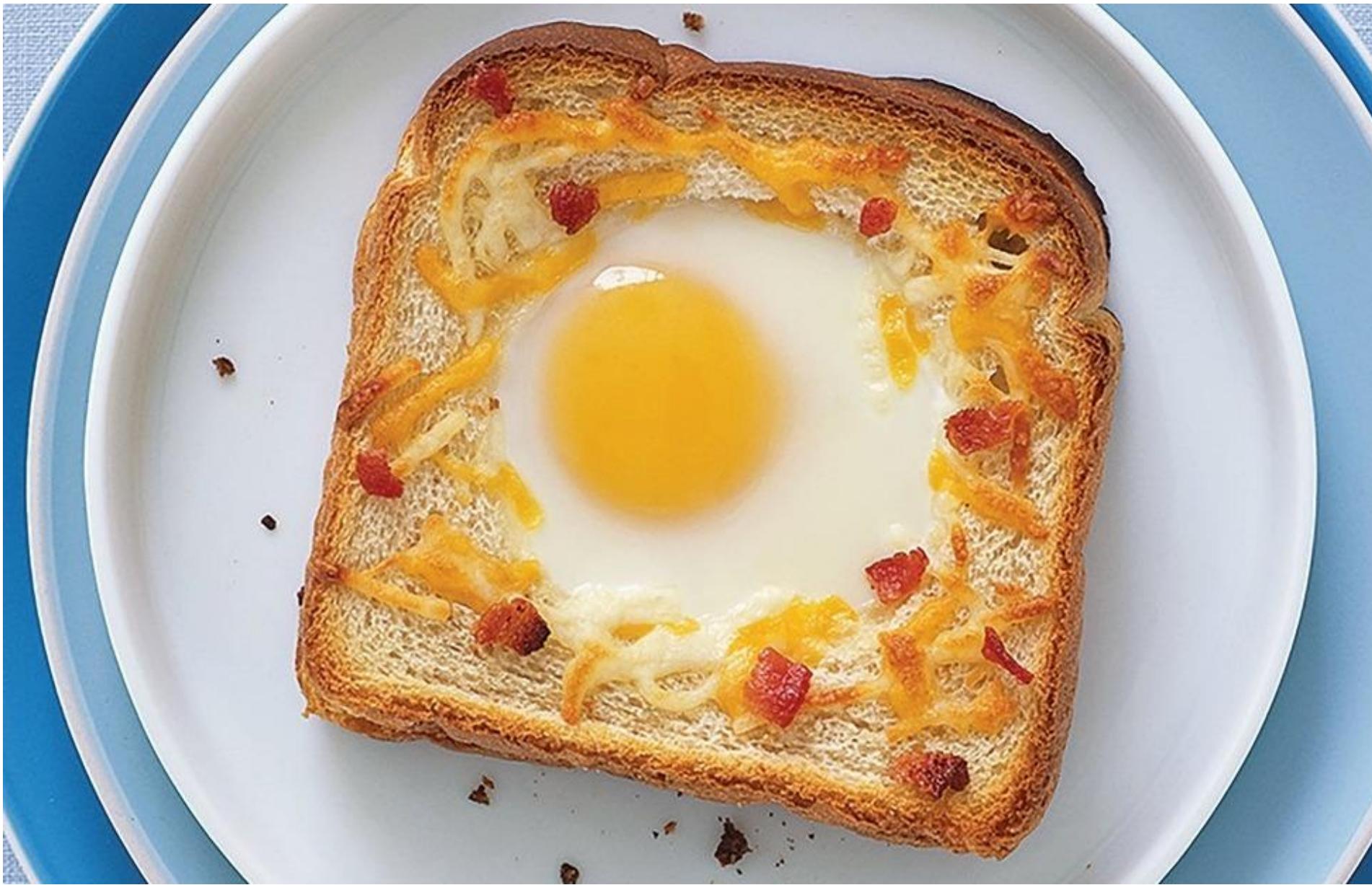
# EXEMPLO

1. Vire a direita
2. Dê 5 passos
3. Desça o degrau
4. Dê 1 passo
5. A porta esta aberta?

Sim: Dê 2 passos e FIM

Não: Abra-a

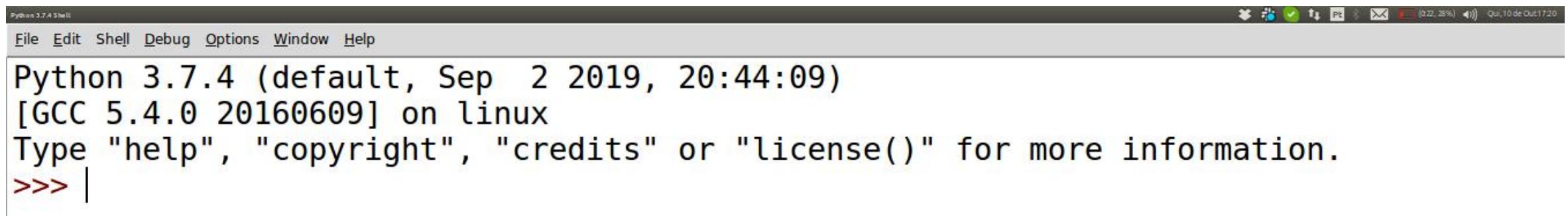
6. Dê 2 passos
7. Fim



# MEU PRIMEIRO PROGRAMA

# Meu primeiro programa

A janela interativa contém um shell Python, que é uma interface texto usada para interagir com a linguagem Python. Daí a nome "janela interativa".



The screenshot shows a terminal window titled "Python 3.7.4 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The status bar at the bottom right shows the date and time: "Qui, 10 de Out 17:20". The main area displays the Python welcome message:

```
Python 3.7.4 (default, Sep 2 2019, 20:44:09)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

O símbolo `>>>` na última linha é chamado de prompt. Aqui é onde você digitará seu código. Vá em frente e digite `print("Hello World")`

# Meu primeiro programa

Quando você pressiona Enter, o Python avalia a expressão, exibe a resposta “Hello World” e, em seguida, aguarda mais comandos

```
Python 3.7.4 (default, Sep 2 2019, 20:44:09)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello World")
Hello World
>>> |
```

A sequência de eventos na janela interativa pode ser descrita como um processo com três etapas:

1. Primeiro, o Python lê o código digitado no prompt.
2. Em seguida, o código é avaliado.
3. Finalmente, a saída é exibida na janela e um novo prompt é exibido.

# Meu primeiro programa

Read- Evaluate- Print - Loop ou REPL .

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (default, Sep 2 2019, 20:44:09)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

# Meu primeiro programa

Digite:

```
>>> import this
```

# The Zen of Python

```
>>> 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!

# The Zen of Python

Bonito é melhor que feio.

Explícito é melhor que implícito.

Simples é melhor que complexo.

Complexo é melhor que complicado.

Linear é melhor do que aninhado.

Esparsos é melhor que denso.

Legibilidade conta.

Casos especiais não são especiais o bastante para quebrar as regras.

Ainda que praticidade vença a pureza.

Erros nunca devem passar silenciosamente.

A menos que sejam explicitamente silenciados.

Dante da ambigüidade, recuse a tentação de adivinhar.

Deveria haver um – e preferencialmente só um – modo óbvio para fazer algo.

Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.

Agora é melhor que nunca.

Embora nunca freqüentemente seja melhor que já.

Se a implementação é difícil de explicar, é uma má idéia.

Se a implementação é fácil de explicar, pode ser uma boa idéia.

Namespaces são uma grande idéia – vamos ter mais dessas!

# Exibindo informações

Para exibir uma informação utilizamos a função print()

```
Python 3.7.4 (default, Sep 2 2019, 20:44:09)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello World")
Hello World
>>> |
```

Eu poderia fazer esse mesmo processo através da criação de um arquivo que é conhecido com o script em Python (File->New File <ctrl+n>).

# Vamos bagunçar as coisas

Todo mundo comete erros - especialmente durante a programação! E caso você ainda não cometeu nenhum erro, vamos começar com isso e estragar algo de propósito para ver o que acontece.

Os “**erros**” cometidos em um programa são chamados de erros e existem dois principais tipos de erros que você experimentará:

1. Erros de sintaxe
2. Erros de Execução

# Vamos bagunçar as coisas

Todo mundo comete erros - especialmente durante a programação! E caso você ainda não cometeu nenhum erro, vamos começar com isso e estragar algo de propósito para ver o que acontece.

Os “**erros**” cometidos em um programa são chamados de erros e existem dois principais tipos de erros que você experimentará:

1. Erros de sintaxe
2. Erros de Execução

# Erros de Sintaxe

Um erro de sintaxe quando você escreve algum código que não é permitido na linguagem Python.

```
>>> print("Hello world)
```

```
SyntaxError: EOL while scanning string literal  
>>>
```

EOL significa End Of Line, portanto, esta mensagem informa que Python leu todo o caminho até o final da linha sem encontrar o final de coisa chamada string literal.

String de Literal é o texto que está entre as “aspas”.

# Erros de Run-time

Os erros não podem ser detectados até que um programa seja executado. Esses erros são conhecidos como erros de tempo de execução porque ocorrem apenas no momento em que um programa é executado.

Para gerar um erro em tempo de execução, altere o código em p1.py para:  
print(Hello, Word)

Você notou como a cor do texto muda para preto quando você removeu as aspas? O IDLE não reconhece mais o Hello, World como uma string.

O que você acha que acontece quando você executa o script? Experimente e Vejo!

# Erros de Run-time



A screenshot of the Python IDLE interface. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The main window shows the code: `print(Hello, World)`. Below the code, the Python interpreter's output is displayed:

```
>>>
==== RESTART: /home/allan/Documentos/Projetos/Curso Pthon/conductor/p1.py ====
Traceback (most recent call last):
  File "/home/allan/Documentos/Projetos/Curso Pthon/conductor/p1.py", line 1, in <
module>
    print(Hello, World)
NameError: name 'Hello' is not defined
```

O que aconteceu? Ao tentar executar o programa Python gerado um erro. Sempre que ocorre um erro, o Python para de executar o programa e exibe o erro na janela interativa do IDLE.

# Erros de Run-time

O texto que é exibido para um erro é chamado de traceback. Ele fornece algumas informações úteis sobre o erro. Que acima traceback nos diz o seguinte:

```
>>>
==== RESTART: /home/allan/Documentos/Projetos/Curso Python/conductor/p1.py ====
Traceback (most recent call last):
  File "/home/allan/Documentos/Projetos/Curso Python/conductor/p1.py", line 1, in <
module>
    print(Hello, World)
NameError: name 'Hello' is not defined
```

- O erro ocorreu na linha 1 do hello\_world.py.
- A linha que gerou o erro foi: print(Hello, World).
- Ocorreu um NameError.
- O erro específico foi o nome 'Hello' não está definido.

# Erros de Run-time

As aspas em torno de Hello, World estão faltando, então o Python não entende que é uma sequência de texto. Em vez disso, Python pensa que Hello e World são os nomes de outra coisa no código.

Como os nomes Hello e World não foram definidos em nenhum lugar, o programa falha.

# Variáveis

No Python, variáveis são nomes que podem ser atribuídos a um valor e usados para referenciar esse valor em todo o seu código. Variáveis são fundamentais à programação por dois motivos:

- 1. As variáveis mantêm os valores acessíveis:** por exemplo, o resultado de alguma operação demorada pode ser atribuída a uma variável para que a operação não precise ser executada toda vez que você precisa usar o resultado.
- 2. Variáveis fornecem contexto de valores:** o número 28 pode significar lotes de diferentes coisas, como o número de alunos em uma classe ou o número de vezes que um usuário acessou um site e assim por diante.

# O operador de atribuição

Os valores são atribuídos a uma variável usando um símbolo especial = chamado de operador de atribuição. Um operador é um símbolo, como = ou +, que realiza alguma operação em um ou mais valores.

Por exemplo, o operador + recebe dois números, um à esquerda do operador e um à direita e os adiciona. Da mesma forma, o = operador pega um valor à direita do operador e o atribui a o nome à esquerda do operador.

Para ver o operador de atribuição em ação, vamos modificar o "Hello, World" que você viu na última seção. Desta vez, usaremos um variável para armazenar algum texto antes de imprimi-lo na tela:

# O operador de atribuição

```
>>> frase="Hello World"  
>>> print(frase)  
Hello World  
>>> |
```

Na primeira linha, uma variável chamada frase é criada e atribuída a valor "Hello World" usando o operador =. A string "Hello World" que foi originalmente usado entre parênteses na função print() foi substituído pela variável **frase**.

A saída Hello World é exibida quando você executa a print(frase) porque o Python procura a variável frase e descobre que foi associado o valor "Olá, mundo".

Se você não executou a frase = "Hello Word" antes de executar print(frase), você teria visto um NameError como quando tentando executar print(Hello Word).

# Igual é igual?

Nome das variáveis são **case-sensitive**, ou seja, Letras maiúsculas são diferentes de letras minúsculas, independente se o nome é igual, portanto, uma variável denominada frase é distinta de uma variável denominada Frase (observe P maiúsculo). Por exemplo, o código a seguir produz um NameError:

```
>>> frase = "Olá Mundo"
>>> print(Frase)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(Frase)
NameError: name 'Frase' is not defined
```

# Regras para nomes de Variáveis

Os nomes de variáveis podem ser tão longos ou curtos quanto você quiser, mas há algumas regras que você deve seguir. Os nomes de variáveis podem conter apenas letras maiúsculas e minúsculas (A – Z, a – z), dígitos (0–9) e Underscore/Sublinhado(\_). No entanto, os nomes de variáveis não podem começar com um dígito.

Por exemplo, frase, string1, \_a1p4a e lista\_de\_nomes são todas variáveis válidas, mas **9vidas NÃO**.

# Nomes descritivos são melhores que nomes curtos

Nomes de variáveis descritivos são essenciais, especialmente para programas. Muitas vezes, nomes descritivos requerem o uso de várias palavras. Não tenha medo de usar nomes longos de variáveis.

No exemplo a seguir, o valor 3600 é atribuído à variável s:

**s=3600**

O que “s” significa? nos poderíamos ser mais preciso com:

**segundos\_por\_hora = 3600**

# Convenções de nomenclatura de variáveis Python

Em muitas linguagens de programação, é comum escrever nomes de variáveis no formato **camelCase** como numEstudantes e listaDeNomes. A primeira letra de cada palavra, exceto a primeira, é maiúscula e todas as outras letras é minúsculo. A justaposição de letras minúsculas e maiúsculas parecem corcundas em um camelo.

No Python, no entanto, é mais comum escrever nomes de variáveis em **snake case** como num\_estudantes e lista\_de\_nomes. Cada letra é minúsculas e cada palavra é separada por um sublinhado.

Embora não exista uma regra rígida que exija que você escreva seu nomes de variáveis com snake case, a prática é codificada em um documento chamado PEP 8, que é amplamente considerado como o guia oficial de estilo para que escreve programas em Python.

Seguir os padrões descritos no PEP 8 garante que o seu código Python é legível por um grande número de programadores Python. Este modo facilita o compartilhamento e a colaboração no código para todos os envolvidos.

# Inspecionando valores com a Janela Interativa

Você já viu como usar `print()` para exibir uma string que possui foi atribuído a uma variável. Há outra maneira de exibir o valor de uma variável quando você estiver trabalhando no shell Python.

Digite o seguinte na janela interativa do IDLE:

```
>>> frase = "Ola"  
>>> frase
```

Quando você pressiona Enter após digitar a frase pela segunda vez, os seguintes a saída é exibida:

```
| 'Ola'
```

Agora digite:

```
>>> print(frase)
```

# Inspecionando valores com a Janela Interativa

Você vê a diferença entre esta saída e a saída com o print?

O que está acontecendo aqui?

Ao digitar **frase** e pressionar Enter, você está dizendo ao Python para inspecionar a variável **frase**.

A saída exibida é um representante útil do valor atribuído à variável.

Nesse caso, a frase recebe "Ola", portanto, a saída é cercada por aspas simples para indicar que a frase é uma sequência (string).

Por outro lado, quando você usa `print()` e uma variável, o Python exibe uma representação mais legível por humanos do valor da variável. Para seqüências de caracteres, ambas as formas de exibição são legíveis por humanos, mas isso não é o caso para todo tipo de valor.

# Inspecionando valores com a Janela Interativa

Às vezes, imprimir e inspecionar uma variável produz o mesma saída:

```
>>> x = 2  
>>> x  
2  
>>> print(x)  
2  
. . .
```

# Inspecionando valores com a Janela Interativa

Às vezes, imprimir e inspecionar uma variável produz o mesma saída:

```
>>> x = 2  
>>> x  
2  
>>> print(x)  
2  
. . .
```

```
>>> x=2  
>>> y='2'  
>>> print(x)  
2  
>>> print(y)  
2  
. . .
```

# Inspecionando valores com a Janela Interativa

Às vezes, imprimir e inspecionar uma variável produz o mesma saída:

```
>>> x = 2
```

```
>>> x
```

```
2
```

```
>>> print(x)
```

```
2
```

```
.
```

```
>>> x=2
```

```
>>> y='2'
```

```
>>> print(x)
```

```
2
```

```
>>> print(y)
```

```
2
```

```
.
```

```
>>> x=2
```

```
>>> y='2'
```

```
>>> x
```

```
2
```

```
>>> y
```

```
'2'
```

```
.
```

# Deixe Anotações Úteis

Os programadores costumam ler o código que escreveram há vários meses e dizem:

**“O que (piii....) isso faz?”**

Mesmo com a variável descritiva nomes, pode ser difícil lembrar por que você escreveu algo que como você fazia quando não olhava há muito tempo.

Para ajudar a evitar esse problema, você pode deixar comentários no seu código.

Comentários são linhas de texto que não afetam a maneira como o script é executado.

Eles ajudam a documentar o que deveria estar acontecendo.

# Como escrever um comentário

---

```
# Isto é um comentário
frase = "Meu programa é show"
print(frase) # Vai exibir o conteúdo de frase - Isso é um comentário em linha

# Primeiro Comentário
# Segundo Comentário
#Uauu que legal, três comentários (está errado - PEP8

frase = "Meu programa é show" # Correto
print(frase)# Errado

...
Comentario Interessante
Utilizando Várias Linhas
Acho que vou utilizar mais disso
...  
...
```

# Vamos Lembrar:

1. **Variáveis** atribuem nomes a valores no seu código usando o operador de atribuição (=)
2. **Erros**, tais erros de sintaxe e erros de tempo de execução, são gerados quando:  
o Python não pode executar seu código. Eles são exibidos nos IDLE's janela interativa na forma de um rastreamento.
3. **Comentários** são linhas de código que não são executadas e servem como documentação para você e outros que precisam ler seu código.

# STRINGS E MÉTODOS DE STRINGS

# Strings e Métodos

Muitos programadores, independentemente de sua especialidade, lidam com texto em uma base diária. Por exemplo, desenvolvedores web trabalham com texto que é obtido entrada de formulários da web. Os cientistas de dados processam o texto para extrair dados e realizar coisas como análise de sentimentos, que podem ajudar a identificar e classifique opiniões em um corpo de texto.

Coleções de texto em Python são chamadas de **strings**. Funções especiais são chamadas métodos são usados para manipular strings, como alterar uma string de minúscula para maiúscula, removendo o espaço em branco de início ou fim da sequência, ou a substituição de partes de uma sequência por texto diferente.

# O que é uma string?

Vamos ver mais detalhadamente o que exatamente é uma string e as várias maneiras de criá-las em Python.

## O tipo de dados String

Strings são um dos tipos de dados fundamentais do Python. O termo tipo de dados refere-se a um tipo de dado que um valor representa. Strings são usadas para representar texto.

# O que é uma string?

```
>>> type("Ola Mundo")
```

A saída <class 'str'> indica que o valor "Ola, mundo" é um instância do tipo de dados str. Ou seja, "Ola, mundo" é uma string.

As strings têm três propriedades que você explorará nos próximos segundos  
ções:

1. Strings contém caracteres, que são letras ou símbolos individuais.
2. As strings têm um comprimento, que é o número de caracteres contido na string.
3. Os caracteres em uma string aparecem em uma sequência, significando cada caractere tem uma posição numerada na string.

# Strings Literais

```
>>> string1="Olá"  
>>> string2="1234"
```

# Tamanho de uma String

```
>>> len(string2)  
4  
,
```

```
>>> letras = 'abc'  
>>> numerosDeLetras = len(letras)  
>>> numerosDeLetras  
3  
>>> |
```

# String em multiplas linhas

```
paragrafo = "Este planeta tem - ou melhor, teve - um problema, que era este: a \
maioria das pessoas que viviam nele estava infeliz por quase todo o tempo. \
Muitas soluções foram sugeridas para esse problema, mas a maioria delas se \
preocupava principalmente com os movimentos de pequenos pedaços de papel verde, \
o que é estranho, porque, no geral, não eram os pequenos pedaços de papel verdes \
estavam infelizes."
print(paragrafo)
```

# Concatenação, indexação e fatiamento

Agora que você sabe o que é uma string e como declarar literais em seu código, vamos explorar algumas das coisas que você pode fazer com as strings.

Nesta seção, você aprenderá sobre três operações básicas de string:

1. Concatenação, que une duas strings
2. Indexação, que obtém um único caractere de uma sequência
3. Fatiar, que obtém vários caracteres de uma sequência de uma só vez

Vamos mergulhar!

# Concatenação de Strings

Duas string podem ser combinadas ou concatenadas, usando o operador +:

```
>>> string1 = "abra"  
>>> string2 = "cadabra"  
>>> string magica = string1+string2  
>>> print(string magica)  
abracadabra  
  
>>> string magica = string1+" "+string2  
>>> string magica  
'abra cadabra'
```

# Índice de String

Cada caractere em uma sequência tem uma posição numerada chamada índice.

Você pode acessar o caractere na posição N-ésima colocando o número entre N entre colchetes ([e]) imediatamente após a string:

```
>>> baton="Saia Justa"  
>>> baton[1]  
'a'
```

baton[1] retorna o caracter na posição 1 de “Saia Justa”. No Python e na maioria das outras linguagens de programação a contagem sempre começa em zero. Para obter o caractere no início de uma string, você precisa acessar o personagem na posição 0.

# Índice de String

O que acontece se eu colocar o valor 12:

```
>>> baton[12]
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    baton[12]
IndexError: string index out of range
```

# Índice de String

S	a	i	a		J	u	s	t	a
0	1	2	3	4	5	6	7	8	9

S	a	i	a		J	u	s	t	a
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

# Fatiamento de String (slice)

Vamos supor que eu queira as primeiras 3 letras da string “Saia Justa”

```
>>> tresletras = baton[0]+baton[1]+baton[2]
>>> tresletras
'Sai'
```

# Fatiamento de String (slice)

Vamos supor que eu queira as primeiras 3 letras da string “Saia Justa”

```
>>> tresletras = baton[0:3]
>>> tresletras
'Sai'
```

# Fatiamento de String (slice)

```
>>> baton[:]
'Saia Justa'
>>>
>>> baton[:5]
'Saia '
>>> baton[5:]
'Justa'
>>> baton[:]
'Saia Justa'
>>> baton[:14]
'Saia Justa'
>>> baton[13:18]
''
>>> baton[-9:-6]
'aia'
>>> baton[ -10:-5]
'Saia '
```

# Fatiamento de String (slice)

```
>>> telefone = "11-97041-9911"
>>> print("**_***** - "+telefone[-5:])
**_***** - 9911
```

# Strings São Imutáveis

Para encerrar esta seção, vamos discutir uma importante propriedade dos objetos de sequência. As strings são **imutáveis**, o que significa que você não pode alterá-las depois de criá-las. Por exemplo, veja o que acontece quando você tenta atribuir uma nova letra a um caractere específico de uma string:

```
>>> palavra = "objetivo"
>>> palavra[0]
'o'
>>> palavra[0] = "u"
Traceback (most recent call last):
  File "<pyshell#54>", line 1, in <module>
    palavra[0] = "u"
TypeError: 'str' object does not support item assignment
```

# Strings São Imutáveis

Para alterar a string "objetivo" para a string "ubjetivo", você pode usar o slice de string para concatenar a letra "u" com tudo, exceto a primeira letra da palavra "objetivo":

```
>>> palavra = "objetivo"  
>>> palavra = "u" + palavra[1:]  
>>> palavra  
'ubjetivo'
```

Primeiro atribua a cadeia "objetivo" à variável palavra variável. Em seguida, concatene a fatia da palavra [1:], que é a string "bjetivo", com a letra "u" para obter a string "ubjetivo". Se você está obtendo um resultado diferente aqui, inclua o caractere: dois-pontos como parte da fatia da string.

# Manipular sequências de caracteres com métodos

As strings são fornecidas com funções especiais chamadas métodos de strings que podem ser usadas para trabalhar e manipular strings. Existem vários métodos de string disponíveis, mas vamos nos concentrar em alguns dos mais usados.

Você aprenderá como:

- Converter uma string em maiúsculas ou minúsculas
- Remover o espaço em branco da string
- Determine se uma sequência começa e termina com certos caracteres

Vamos!

# Convertendo letras em Minuscúlas e Maiuscúlas

```
>>> nome = "Sou pequeno"  
>>> nome.lower()  
'sou pequeno'  
>>> nome = "Sou Pequeno"  
>>> nome.lower()  
'sou pequeno'
```

```
>>> nome="dock é uma empresa maravilhosa"  
>>> nome.upper()  
'DOCK É UMA EMPRESA MARAVILHOSA'
```

# Remover Espaços em Brancos de uma String

Espaço em branco é qualquer caractere impresso como espaço em branco. Isso inclui itens como espaços e “enter”, que são caracteres especiais que movem a saída para uma nova linha.

Às vezes, você precisa remover o espaço em branco do início ou do fim de uma string. Isso é especialmente útil ao trabalhar com seqüências de caracteres provenientes da entrada do usuário, onde caracteres extras de espaço em branco podem ter sido introduzidos por acidente.

Existem três métodos de sequência que você pode usar para remover o espaço em branco de uma sequência:

# Remover Espaços em Brancos de uma String

`rstrip()` - Espaços a Direita

`lstrip()` - Espaços a Esquerda

`strip()` - Ambos

# Remover Espaços em Brancos de uma String

```
>>> nome = "Dock "
>>> nome.rstrip()
'Dock'
>>> nome = " Dock"
>>> nome.lstrip()
'Dock'
>>> nome = " Dock "
>>> nome.strip()
'Dock'
>>> |
```

---

# Remover Espaços em Brancos de uma String

WAT?

```
>>> " Dock" == "Dock "
```

# Remover Espaços em Brancos de uma String

WAT?

```
>>> " Dock" == "Dock "
>>> " Dock".lstrip() == "Dock ".rstrip()
```

# Determine se uma string inicia ou termina com um Sequência Particular

Quando você trabalha com texto, às vezes precisa determinar se uma determinada sequência começa ou termina com determinados caracteres. Você pode usar dois métodos de sequência de caracteres para resolver esse problema:

.startswith () e .endswith ().

Vamos ver um exemplo. Considere a string "Empresa". Veja como você usa .startswith() para determinar se a sequência começa com as letras "e" e "m":

# Determine se uma string inicia ou termina com um Sequência Particular

Quando você trabalha com texto, às vezes precisa determinar se uma determinada sequência começa ou termina com determinados caracteres. Você pode usar dois métodos de sequência de caracteres para resolver esse problema:

.startswith () e .endswith ().

Vamos ver um exemplo. Considere a string "Empresa". Veja como você usa .startswith() para determinar se a sequência começa com as letras "e" e "m":

# Determine se uma string inicia ou termina com um Sequência Particular

```
>>> texto = "Empresa"
>>> texto.startswith("em")
False
>>> texto.startswith("Em")
True

>>> texto = "Empresa"
>>> texto.endswith("Esa")
False
>>> texto.endswith("esa")
True
```

# Métodos de String e Imutabilidade

Strings são Imutáveis, não podem ser alteradas depois de terem sido criados.

A maioria dos métodos que alteram uma string, como `.upper()` e `.lower()`, na verdade retornam cópias da string original com as modificações apropriadas.

Se você não tomar cuidado, isso pode introduzir erros sutis no seu programa.

Experimente isso na janela interativa do IDLE:

```
>>> nome = "Laurito"
>>> nome.upper()
'LAURITO'
>>> nome
'Laurito'
>>> nome = nome.upper()
>>> nome
'LAURITO'
```

# Descobrindo coisas.

Digite:

```
>>> nome = "Dock"  
>>> nome. (pressione <ctrl>+<barra de espaço> ou <tab><tab>)  
>>> dir(nome) -> retorna os atributos de um objeto  
>>> help() - exit ou ctrl+d para sair  
>>> help(print)
```

# Interagindo com o Usuário

Agora que você já viu como trabalhar com métodos de string, vamos tornar as coisas interativas. Vamos aprender como obter alguma entrada de um usuário com a função `input()`. Você escreverá um programa que solicita que o usuário insira algum texto e exibe esse texto novamente em maiúscula.

Para usar a função `input()`, você deve especificar um prompt. O prompt é apenas uma string que você coloca entre os parênteses de `input()`. Pode ser o que você quiser: uma palavra, um símbolo, uma frase - qualquer coisa que seja uma string Python válida.

A função `input()` exibe o prompt e aguarda o usuário digitar algo no teclado. Quando o usuário pressiona Enter, `input()` retorna sua entrada como uma string que pode ser atribuída a uma variável e usada para fazer algo em seu programa.

# Interagindo com o Usuário

Para ver como `input()` funciona, salve e execute o seguinte script:

```
>>> pergunta = "Olá, Como você está?"  
>>> digite = input(pergunta)  
Olá, Como você está?Estou bem  
>>> print("Você disse:",digite)  
Você disse: Estou bem
```

# Interagindo com o Usuário

Outro exemplo:

```
>>> resposta = input("O que você quer falar bem alto?")
O que você quer falar bem alto? eu posso. eu consigo. eu mereço
>>> resposta = resposta.upper()
>>> print("Bem, já que você insiste...",resposta)
Bem, já que você insiste... EU POSSO. EU CONSIGO. EU MEREÇO
```

# Trabalhando com Strings e Números

Quando você obtém a entrada do usuário usando a função `input()`, o resultado é sempre uma string. Há muitos outros momentos em que a entrada é dada a um programa como uma string. Às vezes, essas cadeias contêm números que precisam ser alimentados nos cálculos.

Vamos aprenderá como lidar com cadeias de números. Você verá como as operações aritméticas funcionam em sequências de caracteres e como elas frequentemente levam a resultados surpreendentes. Você também aprenderá como converter entre cadeias e tipos de números.

# Strings e Operadores Aritméticos

Você viu que os objetos de seqüência de caracteres podem conter muitos tipos de caracteres, incluindo números. No entanto, não confunda numerais em uma sequência com números reais. Por exemplo, tente este trecho de código na janela interativa do IDLE:

```
>>> num="2"  
>>> num + num  
'22'
```

# Strings e Operadores Aritméticos

Você viu que os objetos de seqüência de caracteres podem conter muitos tipos de caracteres, incluindo números. No entanto, não confunda numerais em uma sequência com números reais. Por exemplo, tente este trecho de código na janela interativa do IDLE:

```
>>> num="2"      >>> num="12"  
>>> num + num  >>> num*3  
'22'           '121212'
```

# Como um programador python conta uma piada

```
|>>> "NaN"*16+" Batman"
```

# Converter Strings em Números

```
>>> int("2")
```

```
2
```

```
>>> float("2")
```

```
2.0
```

# Converter Strings em Números

```
>>> numero = input("Entre com o número para exibir o seu dobro:")
Entre com o número para exibir o seu dobro:2
>>>dobrodonumero = numero * 2
>>>print(dobrodonumero)
```

# Converter Strings em Números

```
>>> numero = input("Entre com o número para exibir o seu dobro:")  
Entre com o número para exibir o seu dobro:2  
>>>dobrodonumero = numero * 2  
>>>print(dobrodonumero)
```

```
>>> numero = input("Entre com o número para exibir o seu dobro:")  
Entre com o número para exibir o seu dobro:2  
>>>dobrodonumero = float(numero) * 2  
>>>print(dobrodonumero)
```

# Converter Números em Strings

```
>>> numerodepaes = 10  
>>> 'Eu como '+numerodepaes+' pães'
```

# Converter Números em Strings

```
>>> numerodepaes = 10  
>>> 'Eu como '+numerodepaes+' pães'
```

```
>>> numerodepaes = 10  
>>> 'Eu como '+numerodepaes+' pães'  
Traceback (most recent call last):  
  File "<pyshell#12>", line 1, in <module>  
    'Eu como '+numerodepaes+' pães'  
TypeError: can only concatenate str (not "int") to str
```

# Converter Números em Strings

```
>>> numerodepaes = 10
>>> 'Eu como '+str(numerodepaes)+ ' pães'
'Eu como 10 pães'
>>> print('Eu como ',numerodepaes , 'pães ')
Eu como 10 pães
```

# Simplifique suas declarações de print

Suponha que você tenha um string nome = "Zaphod" e dois inteiros cabecas = 2 e armas = 3. Você deseja exibi-los na seguinte linha: Zaphod tem 2 cabeças e 3 braços.

Isso é chamado de **interpolação de string**, que é apenas uma maneira elegante de dizer que você deseja inserir algumas variáveis em locais específicos em uma string.

Você já viu duas maneiras de fazer isso. O primeiro envolve o uso de vírgulas para inserir espaços entre cada parte da string dentro de uma função print():

# Simplifique suas declarações de print

```
>>> nome="Zaphod"
>>> cabeca=2
>>> armas=3
>>> print(nome, "tem", cabeca, " cabeças e", armas, "armas")
Zaphod tem 2 cabeças e 3 armas
>>> print(nome, "tem", str(cabeca), " cabeças e", str(armas), "armas")
Zaphod tem 2 cabeças e 3 armas
>>> print(nome + " tem " + str(cabeca) + " cabeças e " + str(armas) + " armas")
Zaphod tem 2 cabeças e 3 armas
```

Ambas as técnicas produzem código que pode ser difícil de ler. Tentar acompanhar o que ocorre dentro ou fora das aspas pode ser difícil. Felizmente, existe uma terceira maneira de combinar strings: literais de strings formatados, mais conhecidos como f -strings.

A maneira mais fácil de entender as F strings é vê-las em ação. Veja como a string acima se parece quando escrita como uma string f:

# Simplifique suas declarações de print

```
>>> f"{nome} tem {cabeca} cabeças e {armas} armas"  
'Zaphod tem 2 cabeças e 3 armas'
```

Há duas coisas importantes a serem observadas nos exemplos acima:

1. O literal da string começa com a letra f antes das aspas de abertura.
2. Os nomes de variáveis entre colchetes ({} ) são substituídos com seus valores correspondentes sem usar str().

Você também pode inserir expressões Python entre os chaves.

# Simplifique suas declarações de print

As expressões são substituídas pelo resultado na cadeia de caracteres:

```
>>> n = 3  
>>> m = 4  
>>> f"{n} vezes {m} é {n*m}"  
'3 vezes 4 é 12'
```

É uma boa idéia manter as expressões usadas em uma f string o mais simples possível. A inserção de várias expressões complicadas em um literal de cadeia de caracteres pode resultar em código difícil de ler e de manter.

# Simplifique suas declarações de print

As f-strings estão disponíveis apenas no Python versão 3.6 e superior. Nas versões anteriores do Python, o método `.format()` pode ser usado para obter os mesmos resultados. Este método também funciona no Python 3.6 e 3.7. Retornando ao exemplo Zaphod, você pode usar o método `.format()` para formatar a string da seguinte maneira:

```
>>> "{} tem {} cabeças e {} armas".format(nome,cabeca,armas)
'Zaphod tem 2 cabeças e 3 armas'
```

# Buscar um String em uma String

Um dos métodos de string mais úteis é `.find()`. Como o próprio nome indica, você pode usar esse método para encontrar o local de uma string em outra string - geralmente chamada de substring.

Para usar `.find()`, coloque-o no final de uma variável ou literal de cadeia e passe a cadeia que deseja encontrar entre parênteses:

```
>>> frase = "a surpresa está aqui em algum lugar"  
>>> frase.find("surpresa")
```

```
>>> frase = "a surpresa está aqui em algum lugar"  
>>> frase.find("supresa")
```

# Substituir um texto em uma string

```
>>> frase = "A verdade está lá fora"  
>>> frase.replace("fora","dentro")  
'A verdade está lá dentro'  
>>> frase  
'A verdade está lá fora'
```

```
>>> frase = frase.replace("fora","dentro")  
>>> frase  
'A verdade está lá dentro'
```

# Números e Matemática

Você não precisa ser um gênio da matemática para programar bem. A verdade é que poucos programadores precisam saber mais do que álgebra básica.

Obviamente, a quantidade de matemática que você precisa saber depende do aplicativo em que está trabalhando. Em geral, o nível de matemática necessário para trabalhar com sucesso como programador é menor do que você poderia esperar.

Neste capítulo, você aprenderá como:

- Trabalhe com os três tipos de números internos do Python: número inteiro, ponto flutuante e números complexos
- Arredonde números para um determinado número de casas decimais
- Formatar e exibir números em strings

Vamos começar!

# Números inteiros e ponto flutuante

O Python possui três tipos de dados numéricos internos: números inteiros, números de ponto flutuante e números complexos.

# Números inteiros

Um número inteiro é um número inteiro sem casas decimais. Por exemplo, 1 é um número inteiro, mas 1,0 não é. O nome do tipo de dados inteiro é int, que você pode ver com a função type ():

```
>>> type(1)
<class 'int'>
```

Você pode criar um número inteiro simplesmente digitando o número explicitamente ou usando a função int () .

```
>>> int("25")
25
```

Um literal inteiro é um valor inteiro escrito explicitamente no seu código, assim como uma string literal é uma string escrita explicitamente no seu código. Por exemplo, 1 é um literal inteiro, mas int("1") não é. Literais inteiros podem ser escritos de duas maneiras diferentes:

# Números inteiros

```
>>> 1000000
```

```
>>> 1_000_000
```

O primeiro exemplo é direto. Basta digitar 1 seguido de seis zeros. A desvantagem dessa notação é que números grandes podem ser difíceis de ler. Quando você escreve números grandes manualmente, provavelmente agrupa dígitos em grupos de três, separados por vírgula. 1.000.000 é muito mais fácil de ler do que 1000000.

No Python, você não pode usar vírgulas para agrupar dígitos em literais inteiros, mas pode usar um sublinhado (\_). O valor 1\_000\_000 expressa um milhão de maneira mais legível.

# Números Ponto Flutuante

Um número de ponto flutuante, ou aveia, é um número com uma casa decimal. 1.0 é um número de ponto flutuante, assim como -2,75. O nome de um tipo de dados de ponto flutuante é float:

```
>>> type(1.0)
<class 'float'>
>>> float("1.25")
1.25
>>> float(' -1.25 ')
-1.25
>>> 1000.0
1000.0
>>> 1 000.0
1000.0
>>> 1e4
10000.0
```

# Números Ponto Flutuante

Um número de ponto flutuante, ou aveia, é um número com uma casa decimal. 1.0 é um número de ponto flutuante, assim como -2,75. O nome de um tipo de dados de ponto flutuante é float:

```
>>> type(1.0)
<class 'float'>
>>> float("1.25")
1.25
>>> float(' -1.25 ')
-1.25
>>> 1000.0
1000.0
>>> 1 000.0
1000.0
>>> 1e4
10000.0
```

# Números Ponto Flutuante

Um número de ponto flutuante, ou aveia, é um número com uma casa decimal. 1.0 é um número de ponto flutuante, assim como -2,75. O nome de um tipo de dados de ponto flutuante é float:

```
>>> type(1.0)
<class 'float'>
>>> float("1.25")
1.25
>>> float(' -1.25 ')
-1.25
>>> 1000.0
1000.0
>>> 1 000.0
1000.0
>>> 1e4
10000.0
```

# Notação Exponencial

$20 \times 10^{17}$

```
>>> 2e+17  
2e+17  
>>> 2e-17  
2e-17  
>>> 2e+4  
20000.0  
>>> 2e-4  
0.0002  
>>> 2e400  
inf  
>>> -2e400  
-inf  

```

# Operações Aritméticas

```
>>> 1 + 2 # adição  
3  
>>> 1.0 + 2 # adição e conversão de tipo de dado  
3.0  
>>> 1 - 1 # subtração  
0  
>>> -3 # notação para números negativos  
-3
```

# Operações Aritméticas

```
>>> 1 - - 3           >>> 3 * 3 # multiplicação  
4  
>>> 1 -- 3           >>> 3 * 3.  
4  
>>> 1- - 3           >>> 3 * 3.0  
4  
>>> 1 - - 3  
4  
>>> 1 - ( - 3 )  
4  
>>> 1+- 3  
-2
```

# Números Ponto Flutuante

Um número de ponto flutuante, ou aveia, é um número com uma casa decimal. 1.0 é um número de ponto flutuante, assim como -2,75. O nome de um tipo de dados de ponto flutuante é float:

```
>>> type(1.0)
<class 'float'>
>>> float("1.25")
1.25
>>> float(' -1.25 ')
-1.25
>>> 1000.0
1000.0
>>> 1 000.0
1000.0
>>> 1e4
10000.0
```

# Operações Aritméticas

```
>>> 9 / 3 # divisão  
3.0  
>>> 5.0 / 2  
2.5  
>>> int( 9/3)  
3  
>>> 9 // 3 # divisão=pegar a parte inteira  
3  
>>> 5.0//2  
2.0  
>>> -3//2  
-2  
>>> 3%2 # Resto da divisão  
1
```

# Operações Aritméticas

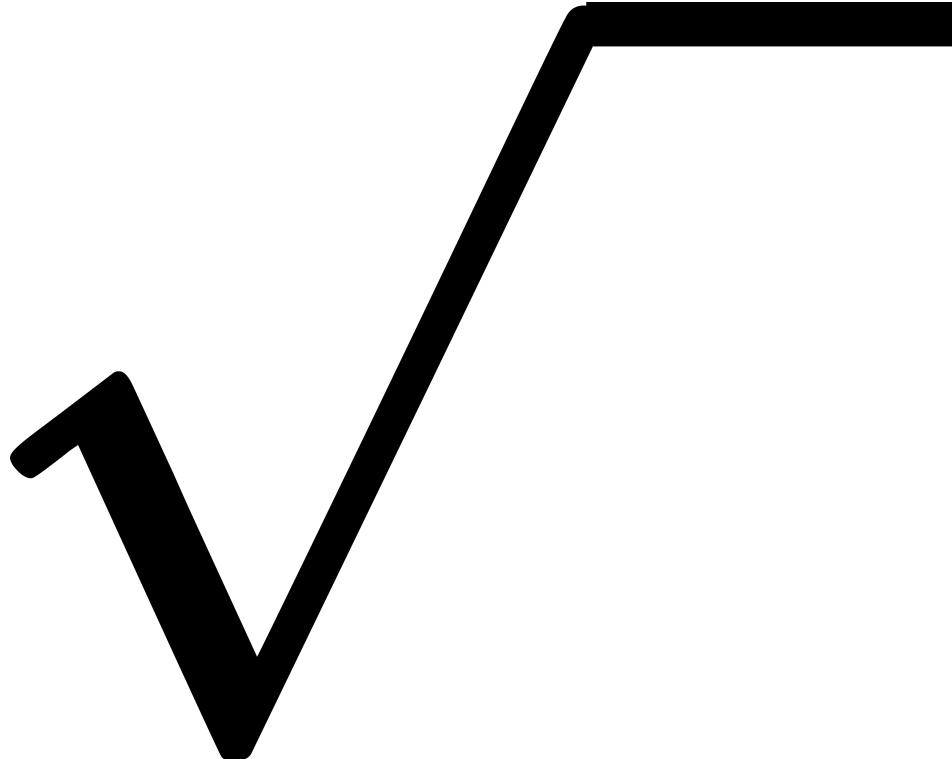
```
>>> 2 ** 2 # Exponenciação 22
4
>>> 2 ** 3
8
>>> 2 ** 4 # 24
16
```

# Números Ponto Flutuante

Um número de ponto flutuante, ou aveia, é um número com uma casa decimal. 1.0 é um número de ponto flutuante, assim como -2,75. O nome de um tipo de dados de ponto flutuante é float:

```
>>> type(1.0)
<class 'float'>
>>> float("1.25")
1.25
>>> float(' -1.25 ')
-1.25
>>> 1000.0
1000.0
>>> 1 000.0
1000.0
>>> 1e4
10000.0
```

# Operações Aritméticas



# Operações Aritméticas



# Operações Aritméticas

O problema:

radix quadratum 9 aequalis 3

# Operações Aritméticas

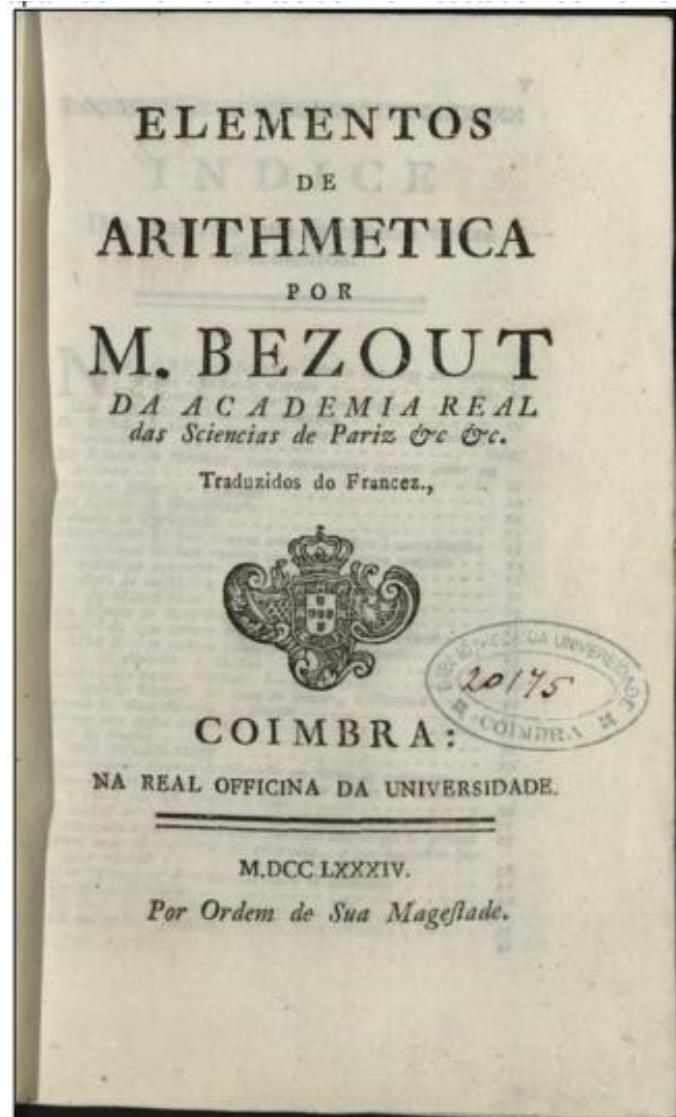
1465

Johannes  
Regiomontanus

**Radix quadratum 9  
(lado do quadrado)**

Carta de  
Regiomontanus  
para Giovanni  
Bianchini  
(Nürnberg  
Stadtbibliothek)

# Operações Aritméticas



Sé o numero proposto constar de hum ou dous algarismos, a sua raiz, em numero inteiro, será algum dos numeros digitos.

I. 2. 3. 4. 5. 6. 7. 8. 9.  
cujos quadrados saõ

1. 4. 9. 16. 25. 36. 49. 64. 81.

Affim v. gr. a raiz quadrada de 72, em numero inteiro, será 8; porque estando 72 entre 64 e 81, a sua raiz tambem se achará entre as raizes dos ditos numeros, que saõ 8, e 9; e por conseguinte será 8 com hum quebrado. Este quebrado porém nunca se pôde determinar exactamente, mas pôde approximar-se continuamente, como mais abaixo se mostrará.

132 A raiz quadrada de hum numero, que não he quadrado perfeito, chama-se numero *surdô irracional*, ou *incommensuravel*.

# Operações Aritméticas



# Programas Mentem?

O que acontecem com essas operações:

```
>>> 0.1 + 0.1  
>>> 0.1 + 0.4  
>>> 0.1 + 0.3  
>>> 0.1 + 0.2
```

# Funções matemáticas e métodos numéricos

O Python possui algumas funções internas que você pode usar para trabalhar com números.

1. `round()`, para arredondar número para algum número de casas decimais
3. `pow()`, exponenciação.
2. `abs()`, para obter o valor absoluto de um número

# Função Round

```
>>> round(2.3)
2
>>> round(2.7)
3
>>> round(2.5)
2
>>> round(3.5)
4
>>> round(1.37)
1
>>> round(3.14159,3)
3.142
>>> round(2.71828,2)
2.72
```

# Função Abs

```
>>> abs(3)
3
>>> abs(-5.0)
5.0
>>>
```

# Função pow

```
>>> pow(2,3)
8
>>> pow(2,2)
4
>>> pow(3,2)
9
>>>
```

# Número inteiro ou Float

```
>>> num = 2.5
>>> num.is_integer()
False
>>>
>>> num = 2.0
>>> num.is_integer()
True
```

# Números com estilo

```
>>> n = 7.125
>>> f"O valor de n é {n}"
'O valor de n é 7.125'
>>> f"O valor de n é {n:.2f}"
'O valor de n é 7.12'
>>>
>>> f"O valor de n é {n:.3f}"
'O valor de n é 7.125'
>>> f"O valor de n é {n:.4f}"
'O valor de n é 7.1250'
```

# Funções e Loops

Funções são os blocos de construção de quase todos os programas Python.

Eles são onde a ação real acontece!

Você já viu como usar várias funções, incluindo `print()`, `len()` e `round()`. Todas essas são funções integradas porque elas são incorporadas à própria linguagem Python.

Você também pode criar funções definidas pelo usuário que executam tarefas específicas.

As funções dividem o código em partes menores e são ótimas para tarefas que um programa usa repetidamente. Em vez de escrever o mesmo código sempre que o programa precisar executar a tarefa, basta chamar a função!

# Funções e Loops

Mas às vezes você precisa repetir algum código várias vezes seguidas, e é aí que entram os loops.

Vamos ver:

- Como criar funções definidas pelo usuário
- Como escrever estruturas de Loops FOR e While
- O que é escopo e por que é importante

Vamos mergulhar!

# O que é uma função, realmente?

Nos usamos funções como `print()` e `len()` para exibir um texto e determinar o comprimento de uma sequência. Mas o que é realmente uma função?

## Funções são valores

Uma das propriedades mais importantes de uma função no Python é que funções são valores e podem ser atribuídas a uma variável.

Na janela interativa do IDLE, inspecione o nome `len` digitando o seguinte no prompt:

```
>>> len  
<built-in function len>
```

# O que é uma função, realmente?

```
>>> len  
<built-in function len>  
>>>  
>>> type(len)  
<class 'builtin_function_or_method'>
```

# Como o Python executa funções?

Agora vamos ver mais de perto como o Python executa uma função.

A primeira coisa a notar é que você não pode executar uma função apenas digitando seu nome. Você deve chamar a função para dizer ao Python para realmente executá-la.

Vamos ver como isso funciona com `len()`:

# Como o Python executa funções?

```
>>> len
<built-in function len>
>>>
>>> len()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: len() takes exactly one argument (0 given)
>>>
```

# Como o Python executa funções?

Neste exemplo, o Python gera um `TypeError` quando `len()` é chamado porque `len()` espera um argumento.

Um argumento é um valor que é passado para a função como entrada.

Algumas funções podem ser chamadas sem argumentos e outras podem receber quantos argumentos você desejar. `len()` requer exatamente um argumento.

Quando uma função é executada, ela retorna um valor como saída. O valor de retorno geralmente - mas nem sempre - depende dos valores de quaisquer argumentos passados para a função.

# Como o Python executa funções?

O processo para executar uma função pode ser resumido em três passos:

1. A função é chamada e todos os argumentos são passados para a função como entrada.
2. A função é executada e alguma ação é executada com os argumentos.
3. A função retorna e a chamada de função original é substituída com o valor de retorno.

# Funções podem ter efeitos colaterais

Você aprendeu a chamar uma função e que eles retornam um valor quando terminam de executar. Às vezes, porém, as funções fazem mais do que apenas retornar um valor.

Quando uma função muda ou afeta algo externo à própria função, diz-se que ela tem um efeito colateral. Você já viu uma função com um efeito colateral: `print()`.

Quando você chama `print()` com um argumento `string`, a `string` é exibida no shell do Python como texto. Mas `print()` não retorna nenhum texto como valor.

Para ver o que `print()` retorna, você pode atribuir o valor de retorno `print()` para uma variável:

# Funções podem ter efeitos colaterais

```
>>> retorno = print("O que eu retorno?")
O que eu retorno?
>>> retorno
>>>
>>> type(retorno)
<class 'NoneType'>
>>>
>>> print(retorno)
None
```

# Minhas funções.

Ao escrever programas mais longos e mais complexos, você pode achar que precisa usar as mesmas poucas linhas de código repetidamente. Ou talvez você precise calcular a mesma fórmula com valores diferentes várias vezes no seu código.

Você pode ser tentado a copiar e colar código semelhante a outras partes do seu programa e modificá-lo conforme necessário, mas isso geralmente é uma má idéia!

O código repetitivo pode ser um pesadelo para manter. Se você encontrar algum erro em algum código que foi copiado e colado em todo o lugar, você precisará aplicar a correção em todos os lugares em que o código foi copiado. Isso dá muito trabalho e você pode perder um lugar!

Nesta seção, você aprenderá a definir suas próprias funções para evitar se repetir quando precisar reutilizar o código. Vamos!

# Anatomia de uma Função

Cada função tem duas partes:

1. A **assinatura da função** define o nome da função e todas as entradas que ela espera.
2. O **corpo da função** contém o código que é executado toda vez que a função é usada.

Vamos começar escrevendo uma função que recebe dois números como entrada e retorna seu produto. Veja como essa função pode ser, com a assinatura, o corpo e a declaração de retorno identificados com os comentários:

# Minha primeira função

```
def minha_funcao(): # Assinatura da Função  
    # Corpo da função  
    print("minha primeira função")  
  
minha_funcao()
```

# Minha primeira função

```
def soma(a,b): # Assinatura da Função  
    # Corpo da Função  
    soma_numeros = a + b  
    print("A soma de a + b é:",soma_numeros)
```

```
soma(1,2)
```

# Minha primeira função

```
def somar_numeros(a,b): # Assinatura da Função  
    # Corpo da Função  
    soma = a + b  
    return soma  
  
resultado = somar_numeros(1,2)  
print("O retorno da função somar_numeros(1,2) é:", resultado)
```

# Executar em círculos

Uma das grandes coisas dos computadores é que você pode fazê-los fazer a mesma coisa repetidamente, e eles raramente reclamam ou se cansam.

Um loop é um bloco de código que é repetido várias vezes ou um número especificado de vezes ou até que alguma condição seja atendida. Existem dois tipos de loops no Python: loops while e for loops.

# O loop WHILE

O loop while repetem uma seção do código enquanto alguma condição é verdadeira.

Há duas partes para o loop while:

1. A instrução while começa com a palavra-chave while, seguida por uma condição de teste e termina com dois pontos (:).
2. O corpo do loop contém o código que é repetido a cada etapa do laço. Cada linha é recuada quatro espaços.

Quando um loop while é executado, o Python avalia a condição de teste e determina se é verdadeira ou falsa. Se a condição de teste for verdadeira, o código no corpo do loop será executado. Caso contrário, o código no corpo é ignorado e o restante do programa é executado.

# O loop WHILE

Quando um loop while é executado, o Python avalia a condição de teste e determina se é verdadeira ou falsa. Se a condição de teste for verdadeira, o código no corpo do loop será executado. Caso contrário, o código no corpo é ignorado e o restante do programa é executado.

Se a condição de teste for verdadeira e o corpo do loop for executado, uma vez que o Python chegue ao final do corpo, ele retornará à instrução while e reavaliará a condição de teste. Se a condição de teste ainda for verdadeira, o corpo será executado novamente. Se for falso, o corpo é ignorado.

Esse processo se repete repetidamente até que a condição de teste falhe, fazendo com que o Python faça um loop sobre o código no corpo do loop while.

# 0 loop WHILE

```
>>> n = 1
>>> while n < 5:
...     print(n)
...     n = n + 1
...
1
2
3
4
```

# O loop WHILE.

Primeiro, o número inteiro 1 é atribuído à variável **n**. Em seguida, um loop while é criado com a **condição de teste n < 5**, que verifica se o valor de n é ou não menor que 5.

Se n for menor que 5, o corpo do loop será executado. Existem duas linhas de código no corpo do loop. Na primeira linha, o valor de n é impresso na tela e, em seguida, n é incrementado em 1 na segunda linha.

# O loop WHILE.

Primeiro, o número inteiro 1 é atribuído à variável **n**. Em seguida, um loop while é criado com a **condição de teste n < 5**, que verifica se o valor de n é ou não menor que 5.

Se n for menor que 5, o corpo do loop será executado. Existem duas linhas de código no corpo do loop. Na primeira linha, o valor de n é impresso na tela e, em seguida, n é incrementado em 1 na segunda linha.

# O loop WHILE.

```
>>> num = float(input("Digite um número positivo:"))
Digite um número positivo:2
>>> while num <= 0:
...     print("O número não é positivo")
...     num = float(input("Digite um número positivo:"))
...
```

# O loop FOR.

Um loop for executa uma seção do código uma vez para cada item em uma coleção de itens. O número de vezes que o código é executado é determinado pelo número de itens na coleção.

Como sua contraparte while, o loop for tem duas partes principais:

1. A instrução for começa com a palavra-chave for, seguida por uma expressão de associação e termina em dois pontos (:).
2. O corpo do loop contém o código a ser executado em cada etapa do loop e possui quatro espaços recuados.

Vamos ver um exemplo. O seguinte para loop imprime cada letra da string "Python" uma de cada vez:

# O loop FOR.

```
>>> for valor in range(0,6):  
...     print(valor)  
...  
0  
1  
2  
3  
4  
5
```

# Lógica condicional e fluxo de controle

Vamos aprender a escrever programas que executam ações diferentes com base em condições diferentes usando lógica condicional.

Emparelhada com funções e loops, a lógica condicional permite escrever programas complexos que lidam com muitas situações diferentes.

- Compare os valores de duas ou mais variáveis
- Escreva instruções if para controlar o fluxo de seus programas
- Manipule erros com tentativa e exceto
- Aplique lógica condicional para criar simulações simples

# Comparando Valores (operadores relacionais)

A lógica condicional é baseada na execução de ações diferentes, dependendo se alguma expressão, chamada condicional, é verdadeira ou não falso. Essa ideia não é específica para computadores. Os seres humanos usam lógica condicional o tempo todo para tomar decisões.

$a > b$

$a < b$

$a \geq b$

$a \leq b$

$a \neq b$

$a == b$

# Comparando Valores (operadores relacionais)

A lógica condicional é baseada na execução de ações diferentes, dependendo se alguma expressão, chamada condicional, é verdadeira ou não falso. Essa ideia não é específica para computadores. Os seres humanos usam lógica condicional o tempo todo para tomar decisões.

$a > b$

$a < b$

$a \geq b$

$a \leq b$

$a \neq b$

$a == b$

# Comparando Valores (operadores relacionais)

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>>
>>> 1 == 1
True
>>> 3 > 5
False
```

```
>>> "a" == "a"
True
>>> "a" == "b"
False
>>> "a" < "b"
True
>>> "a" > "b"
False
>>>
```

# Adicionando um pouco de lógica (operadores lógicos)

O operador **and**.

Considere a seguinte declaração:

1. Gatos tem 4 patas
2. Gatos tem caudas.

Ambas as declarações são verdadeiras.

# Adicionando um pouco de lógica (operadores lógicos)

O operador **and**.

```
>>> 1 < 2  
True  
>>> 1 < 2 and 3 < 4  
True  
>>> 2 < 1  
False  
>>> 2 < 1 and 4 < 3  
False
```

# Adicionando um pouco de lógica (operadores lógicos)

O operador **and**.

```
>>> 1 < 2 and 4 < 3  
False  
>>> 2 < 1 and 3 < 4  
False
```

# Adicionando um pouco de lógica (operadores lógicos)

O operador **and**.

```
>>> True and True  
True  
>>> True and False  
False  
>>> False and True  
False  
>>> False and False  
False
```

# Adicionando um pouco de lógica (operadores lógicos)

O operador **or**.

```
>>> 1 < 2 or 3 < 4
```

```
True
```

```
>>> 2 < 1 or 4 < 3
```

```
False
```

```
>>> 1 < 2 or 4 < 3
```

```
True
```

```
>>> 2 < 1 or 3 < 4
```

```
True
```

# Adicionando um pouco de lógica (operadores lógicos)

O operador **or**.

```
>>> True or True  
True  
>>> True or False  
True  
>>> False or True  
True  
>>> False or False  
False
```

# Adicionando um pouco de lógica (operadores lógicos)

O operador **not**.

```
>>> not True  
False  
>>>  
>>> not False  
True
```

# Controle o fluxo do seu programa

Agora que podemos comparar valores entre si com comparadores booleanos e criar instruções condicionais complexas com operadores lógicos, podemos adicionar alguma lógica ao nosso código que executa ações diferentes para condições diferentes.

# A declaração if

```
>>> if 2 + 2 == 4:  
...     print("2 + 2 = 4")  
...  
2 + 2 = 4  
>>>
```

# A declaração if

```
>>> nota = 95
>>> if nota >= 70:
...     print("Você foi aprovado")
...
Você foi aprovado
```

# A declaração if

```
nota = 40
if nota >= 70:
    print("Você passou")

if nota < 70:
    print("Você não passou")

print("Obrigado")
```

# A palavra reservada else

```
nota = 40
if nota >= 70:
    print("Você passou")
else:
    print("Você não passou")
print("Obrigado")
```

# A palavra reservada elif

```
nota = 80  
  
if nota >= 70:  
    print("Você passou com A")  
elif nota >= 60:  
    print("Você passou com B")  
elif nota >= 50:  
    print("Você passou com C")  
else:  
    print("Você não passou")  
  
print("Obrigado")
```

# Exercício:

Faça um program que entre com a quantidade Gols do Time A e do Time B e que exiba quem venceu e perdeu a partida ou se houve empate entre as equipes.

# Declaração Break, Continue e Pass

```
for letra in "string":  
    if letra=="r":  
        break  
    print(letra)
```

# Declaração Break, Continue e Pass

```
for letra in "string":  
    if letra=="r":  
        continue  
    print(letra)
```

# Declaração Break, Continue e Pass

```
empresa = "dock"  
for letra in empresa:  
    pass
```

# For com Else

```
for n in range(3):
    password = input("Senha: ")
    if password == "teste":
        break
    print("Senha incorreta.")
else:
    print("Atividade Suspeita no Sistema")
```

# Entendendo os Erros

## ValueError

Um ValueError ocorre quando uma operação encontra um valor inválido. Por exemplo, tentar converter a sequência "não é um número" em um número inteiro resulta em um ValueError.

## TypeError

Um TypeError ocorre quando uma operação é executada em um valor do tipo errado. Por exemplo, tentar adicionar uma string e um número inteiro resultará em um TypeError.

## NameError

Um NameError ocorre quando você tenta usar um nome de variável que ainda não foi definido.

# Entendendo os Erros

## **ZeroDivisionError**

Um ZeroDivisionError ocorre quando o divisor em uma operação de divisão é 0.

## **OverflowError**

Um OverflowError ocorre quando o resultado de uma operação aritmética é muito grande. Por exemplo, tentar calcular o valor 2.0 elevado a 1\_000\_000 resulta em um OverflowError:

# Entendendo os Erros

```
>>> int("Não é um número")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'Não é um
número'
```

# Entendendo os Erros

```
>>> int("Não é um número")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'Não é um
número'
```

```
>>> "1" + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

# Entendendo os Erros

```
>>> print(variavel_nao_existe)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'variavel_nao_existe' is not defined
```

# Entendendo os Erros

```
>>> print(variavel_nao_existe)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'variavel_nao_existe' is not defined
```

```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

# Entendendo os Erros

```
>>> pow(2.0,1_000_000)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: (34, 'Numerical result out of range')
```

# Tratando Erros com Try and Except:

```
try:  
    numero = int(input("Entre com um número inteiro:"))  
except ValueError:  
    print("Esse não é um número inteiro")
```

```
def dividir(num1, num2):  
    try:  
        print(num1/num2)  
    except (TypeError, ZeroDivisionError):  
        print("Um problema foi encontrado")  
  
dividir(1,0)
```

# O módulo RANDOM

O Python fornece várias funções para gerar números aleatórios no módulo aleatório.

Um módulo é uma coleção de códigos relacionados. A biblioteca padrão do Python é uma coleção organizada de módulos que você pode importar para seu próprio código para resolver vários problemas.

Para importar o módulo aleatório, digite o seguinte na janela interativa do IDLE:

# O módulo RANDOM

O Python fornece várias funções para gerar números aleatórios no módulo aleatório.

Um módulo é uma coleção de códigos relacionados. A biblioteca padrão do Python é uma coleção organizada de módulos que você pode importar para seu próprio código para resolver vários problemas.

Para importar o módulo aleatório, digite o seguinte na janela interativa do IDLE:

```
>>> import random
```

# O módulo RANDOM

Agora podemos usar funções do módulo aleatório em nosso código. Por exemplo, o randint() possui dois parâmetros necessários chamados a e b, e retorna um número inteiro aleatório maior ou igual a e menor ou igual a b. Ambos a e b devem ser inteiros.

Por exemplo, o código a seguir produz um número inteiro aleatório entre e 10:

```
>>> import random  
>>> random.randint(1,10)  
10
```

# O módulo RANDOM

Além disso, cada número inteiro entre a e b é igualmente provável que seja retornado por randint (). Portanto, para randint (1, 10), cada número inteiro entre 1 e 10 tem 10% de chance de ser retornado. Para randint (0, 1), há uma chance de 50% de retorno de 0.

# Tuplas, Listas e Dicionários

Até agora, você trabalha com tipos de dados fundamentais como str, int e float. Muitos problemas do mundo real são mais fáceis de resolver quando tipos simples de dados são combinados em estruturas de dados mais complexas.

Uma estrutura de dados modela uma coleção de dados, como uma lista de números, uma linha em uma planilha ou um registro em um banco de dados. Modelar os dados com os quais seu programa interage usando a estrutura de dados correta geralmente é a chave para escrever código simples e eficaz.

O Python possui três estruturas de dados internas: tuplas, listas e dicionários.

- Como trabalhar com tuplas, listas e dicionários
- O que é imutabilidade e por que é importante
- Quando usar diferentes estruturas de dados

# Tuplas são sequências imutáveis

Uma sequência é uma lista ordenada de valores. Cada elemento de uma sequência recebe um número inteiro, chamado índice, que determina a ordem em que os valores aparecem. Assim como as strings, o índice do primeiro valor em uma sequência é 0.

Por exemplo, as letras do alfabeto formam uma sequência cujo primeiro elemento é A e o último elemento é Z. Strings também são sequências. A cadeia "Python" possui seis elementos, começando com "P" em índice 0 e "n" no índice 5.

Alguns exemplos reais de sequências incluem os valores emitidos por um sensor a cada segundo, a sequência das notas dos testes de um aluno ou a sequência dos valores diários de estoque de alguma empresa durante um período de tempo.

# O que é uma Tupla

A palavra tupla vem da matemática, onde é usada para descrever uma sequência ordenada finita de valores.

Normalmente, matemáticos escrevem tuplas listando cada elemento, separado por vírgula, dentro de um par de parênteses.  $(1, 2, 3)$  é uma tupla contendo três números inteiros.

As tuplas são ordenadas porque seus elementos aparecem de maneira ordenada. O primeiro elemento de  $(1, 2, 3)$  é 1, o segundo elemento é 2 e o terceiro é 3.

O Python empresta o nome e a notação para tuplas da matemática.

# Criando uma Tupla

Existem algumas maneiras de criar uma tupla no Python. Vamos cobrir dois deles:

1. Literais de tupla
2. A tupla () embutida

# Criando uma Tupla

Existem algumas maneiras de criar uma tupla no Python. Vamos cobrir dois deles:

1. Literais de tupla
2. A tupla () embutida

```
>>> minha_primeira_tupla = (1,2,3)
>>> type(minha_primeira_tupla)
<class 'tuple'>
```

# Criando uma Tupla

```
>>> tupla_vazia = ()  
>>> type(tupla_vazia)  
<class 'tuple'>
```

```
>>> x = (1) # o que é isto?
```

# Criando uma Tupla

```
>>> x = (1,) # o que é isto?
```

```
>>> tuple("Python")
('P', 'y', 't', 'h', 'o', 'n')
```

```
>>> x = (1,2,3)
>>> type(x)
<class 'tuple'>
>>> x[1]
```

# Tuplas são imutaveis

```
>>> x = (1,2,3)
>>> x[0] = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

# Tuplas são iteráveis:

```
vogais = ("a", "e", "i", "o", "u")
for vogal in vogais:
    print(vogal.upper())
```

# Embalagem e desembalagem de tupla:

```
>>> coordenadas = 4.21, 9.29  
>>> type(coordenadas)  
<class 'tuple'>
```

```
>>> coordenadas = 4.21, 9.29  
>>> type(coordenadas)  
<class 'tuple'>  
>>>  
>>> x,y = coordenadas  
>>> x  
4.21  
>>> y  
9.29
```

# Verificando a Existências de Valores com o “in”

```
>>> vogais = ("a", "e", "i", 'o', 'u')
>>> 'e' in vogais
True
>>> 'f' in vogais
False
```

# Retornando Multiplos valores a partir de uma função

```
>>> def soma_subtrai(num1,num2):  
...     return (num1 + num2, num1 - num2)  
...  
>>> soma_subtrai(3,2)  
(5, 1)
```

# Listas são sequências mutáveis

A estrutura de dados da lista é outro tipo de sequência no Python. Assim como cadeias de caracteres e tuplas, as listas contêm itens indexados por números inteiros, começando com 0.

Aparentemente, as listas parecem e se comportam muito como tuplas. Você pode usar notação de índice e fatiamento com listas, verificar a existência de um elemento usando `in` e iterar sobre listas com um loop `for`.

Ao contrário das tuplas, no entanto, as listas são mutáveis, o que significa que você pode alterar o valor em um índice, mesmo após a criação da lista.

# Criando Listas

```
>>> cores = ["vermelho", "amarelo", "verde", "azul"]
>>> type(cores)
<class 'list'>
>>>
>>> cores
['vermelho', 'amarelo', 'verde', 'azul']
>>>
```

# Criando Listas

```
>>> cores_tupla = ('vermelho', 'amarelo', 'verde', 'azul')
>>> type(cores_tupla)
<class 'tuple'>
>>> cores_lista = list(cores_tupla)
>>> type(cores_lista)
<class 'list'>
>>> cores_lista
['vermelho', 'amarelo', 'verde', 'azul']
```

# Criando Listas

```
>>> doces = 'sonho,balas,pirulitos'  
>>> doces_lista = doces.split(",")  
>>> doces_lista  
['sonho', 'balas', 'pirulitos']
```

# Operações com Listas

```
>>> numeros = [1,2,3,4,5]
>>> numeros[1] # Acessando elementos pelo indice
2
>>> numeros[1:3] # Acessando com o recurso de slice
[2, 3]
>>> "João" in numeros
False
```

# Operações com Listas

```
>>> for numero in numeros:  
...     if numero % 2 == 0:  
...         print(f"{numero} é par")  
...     else:  
...         print(f"{numero} é impar")  
...  
1 é impar  
2 é par  
3 é impar  
4 é par  
5 é impar
```

# Operações com Listas

```
>>> cores = ['vermelho', 'amarelo', 'verde', 'azul']
>>> cores[0]
'vermelho'
>>> cores[0] = 'laranja'
>>> cores
['laranja', 'amarelo', 'verde', 'azul']
```

# Operações com Listas

```
>>> cores[1:3] = ['roxo', 'rosa']
>>> cores
['laranja', 'roxo', 'rosa', 'azul']
```

# Listas - Métodos para adicionar e Remover elementos

```
>>> cores = ['vermelho', 'amarelo', 'verde', 'azul']
>>> cores.insert(1,'laranja') # insere na 2o. Posição
>>> cores
['vermelho', 'laranja', 'amarelo', 'verde', 'azul']
```

```
>>> cor = cores.pop(3)
>>> cor
'verde'
>>> cores
['vermelho', 'laranja', 'amarelo', 'azul']
```

# Listas - Métodos para adicionar e Remover elementos

```
>>> cores.append("indigo")
>>> cores
['vermelho', 'laranja', 'amarelo', 'azul', 'indigo']
```

```
>>> cores.extend(['violeta','ultraazul'])
>>> cores
['vermelho', 'laranja', 'amarelo', 'azul', 'indigo', 'violeta',
 , 'ultraazul']
>>> █
```

# Aninhando, copiando e classificando tuplas e listas

Agora que você aprendeu o que são tuplas e listas, como criá-las e algumas operações básicas com elas, vejamos mais três conceitos:

1. Aninhamento
2. Copiando
3. Classificação

# Listas de aninhamento e tuplas

Listas e tuplas podem conter valores de qualquer tipo. Isso significa que listas e tuplas podem conter listas e tuplas como valores. Uma lista aninhada, ou tupla aninhada, é uma lista ou tupla que está contida como um valor em outra lista ou tupla.

```
>>> dois_por_dois = [[1,2],[3,4]]  
>>> len(dois_por_dois)  
2  
>>> dois_por_dois[0]  
[1, 2]  
>>> dois_por_dois[1]  
[3, 4]  
>>>  
>>> dois_por_dois[1][0]  
3
```

# Copiando uma lista

```
>>> animais = ['leao','tigre','fuinha']
>>> animais_copia = animais
>>> animais_copia.append('aguia')
>>> animais
```

# Copiando uma lista de forma independente.

```
>>> animais = ['leao', 'tigre', 'fuinha']
>>> animais_copia = animais[:]
>>> animais_copia.append('aguia')
>>> animais_copia
['leao', 'tigre', 'fuinha', 'aguia']
>>> animais
```

# Ordenando Listas

```
>>> cores = ['vermelho', 'amarelo', 'verde', 'azul']
>>> cores.sort()
>>> cores
```

```
>>> numeros = [1,10,3,2]
>>> numeros.sort()
>>> numeros
```

```
>>> cores.reverse()
>>> numeros.reverse()
>>>
>>> numeros.sort()
>>> cores.sort()
>>>
>>> numeros.sort(reverse=True)
>>> cores.sort(reverse=True)
```

# Armazenando Relacionamentos em Dicionários

Uma das estruturas mais úteis do Python é o dicionário. Como listas e tuplas, os dicionários são usados como uma maneira de armazenar uma coleção de objetos.

No entanto, em vez de armazenar objetos em uma sequência, os dicionários mantêm informações em pares de dados chamados pares de valores-chave.

Em termos gerais, a chave em um par de valores-chave pode ser considerada como "dar um nome a" ou "identificar" a parte do valor do par. Por exemplo, você pode usar um dicionário para armazenar dados de funcionários. Os dados armazenados podem ser um ID de funcionário, um email e uma posição.

Os pares de valores-chave para uma instância deste dicionário podem ser algo como:

# Armazenando Relacionamentos em Dicionários

```
empregado = {  
    "id" : 1702,  
    "email" : "wfreitas@empresa.com.br",  
    "posicao" : "CAO",  
}
```

# Armazenando Relacionamentos em Dicionários

Uma vantagem de um dicionário é que ele contextualiza os valores que ele armazena.

Portanto, para o dicionário, o emprega["id"] recupera o ID, o empregado["email"] o email e o empregado ["posicao"] a posição do funcionário. Isso é inherentemente facilita a compreensão do seu código, mesmo que exija um pouco mais de digitação.

Outra vantagem do dicionário é que procurar o valor de uma chave é muito mais rápido, em geral, do que encontrar um item em uma lista por índice.

Quando você solicita o valor vinculado a uma chave, um dicionário pode pular direto para esse valor e obter seus dados.

# Armazenando Relacionamentos em Dicionários

Por outro lado, quando você solicita um valor em uma lista ou tupla usando seu índice, o Python precisa percorrer a lista desde o início até atingir esse índice.

Pense em um dicionário como uma espécie de pasta usada para organizar papéis para as disciplinas escolares. Os assuntos são separados por guias com o nome do assunto escrito. Quando você deseja encontrar os papéis para a aula de matemática, agarra a aba marcada "Matemática" e pula direto para os papéis.

Por outro lado, uma lista é como um fichário sem guias. Os documentos da turma estão lá, mas você não sabe por onde começar. Então você começa do início e vira as páginas até encontrar os papéis certos. Por esse motivo, os dicionários formam uma boa estrutura para armazenar dados que não possuem uma ordem natural ou que precisam ser recuperados rapidamente.

# Operações com Dicionários

```
>>> agenda = {'Mike Jones': '281-330-8004', 'Jenny': '867-5309',
... 'Destiny': '900-783-3369', 'Obama': '202-456-1414'}
>>>
>>> del(agenda['Destiny'])
>>>
>>> agenda
{'Mike Jones': '281-330-8004', 'Jenny': '867-5309', 'Obama': '202-456-1414'}
>>>
>>> agenda.keys()
dict_keys(['Mike Jones', 'Jenny', 'Obama'])
>>> agenda.values()
dict_values(['281-330-8004', '867-5309', '202-456-1414'])
>>>
>>> for contato in agenda:
...     print(contato,agenda[contato])
...
Mike Jones 281-330-8004
Jenny 867-5309
Obama 202-456-1414
```

# Operações com Dicionários

```
>>> 'Jenny' in agenda  
True  
>>> 'jenny' in agenda  
False  
>>> '555-0199' in agenda  
False  
>>> '867-5309' in agenda  
False
```

# Operações com Dicionários

```
>>> agenda[42] = 'GMG'  
>>> agenda  
{'Mike Jones': '281-330-8004', 'Jenny': '867-5309', 'Obama': '202-456-1414', 42: 'GMG'}  
■
```

```
>>> agenda[[1,2,3]] = "Sera que pode?"
```

```
>>> agenda = {"Jenny": {"cell": "555-0199", "home": "867-5309"},  
... "Mike Jones": {"home": "281-330-8004"},  
... "Destiny": {"work": "900-783-3369"}  
... }  
>>>  
>>> agenda
```

# Operações com Dicionários

```
>>> agenda['Jenny']
{'cell': '555-0199', 'home': '867-5309'}
>>> agenda['Jenny']['cell']
'555-0199'
```

```
>>> dic_simples = dict(nome='jose',idade='32',posicao=3)
>>> dic_simples
```

```
>>> dic_simples = dict([('nome','jose'),('idade',32),('posicao',3)])
>>> dic_simples
```

# Conversão entre Listas, Tuplas e Dicionários

```
>>> a = [1,2,3]
>>> type(a)
<class 'list'>
>>> b = tuple(a)
>>> b
(1, 2, 3)
>>> type(b)
<class 'tuple'>
>>>
>>> placar = {"David": 100, "Dan": 150, "Flavio": 175}
>>> list(placar)
['David', 'Dan', 'Flavio']
>>> tuple(placar)
('David', 'Dan', 'Flavio')
```

# Conversão entre Listas, Tuplas e Dicionários

```
>>> list(placar.items())
[('David', 100), ('Dan', 150), ('Flavio', 175)]
>>> tuple(placar.items())
 (('David', 100), ('Dan', 150), ('Flavio', 175))
```

# Trabalhando com Arquivos - Criar

```
>>> arquivo = open("ola.txt","w") # iniciando um novo arquivo  
>>> arquivo.writelines("Este é o meu primeiro arquivo")  
>>> arquivo.close()
```

# Trabalhando com Arquivos - Criar

```
>>> arquivo = open("ola.txt","w") # iniciando um novo arquivo  
>>> arquivo.writelines("Este é o meu primeiro arquivo")  
>>> arquivo.close()
```

```
>>> arquivo = open("ola.txt","w") # iniciando um novo arquivo  
>>> linhas = ['linha 1','linha 2','linha3']  
>>> arquivo.writelines(linhas)  
>>> arquivo.close()
```

# Trabalhando com Arquivos - Criar

```
>>> arquivo = open("ola.txt","w") # iniciando um novo arquivo  
>>> arquivo.writelines("Este é o meu primeiro arquivo")  
>>> arquivo.close()
```

```
>>> arquivo = open("ola.txt","w") # iniciando um novo arquivo  
>>> linhas = ['linha 1','linha 2','linha3']  
>>> arquivo.writelines(linhas)  
>>> arquivo.close()
```

```
>>> arquivo = open("ola.txt","w") # iniciando um novo arquivo  
>>> linhas = ['linha 1','\nlinha 2','\nlinha 3']  
>>> arquivo.writelines(linhas)  
>>> arquivo.close()
```

# Trabalhando com Arquivos - Criar

```
>>> arquivo = open("ola.txt","a") # carregando um arquivo  
>>> arquivo.writelines('\nNova linha')  
>>> arquivo.close()
```

# Trabalhando com Arquivos - Ler

```
>>> meu_arquivo = open("ola.txt","r")
>>> print(meu_arquivo.readlines())
['linha 1\n', 'linha 2\n', 'linha 3\n', 'Nova linha\n', 'Nova linha']
>>> meu_arquivo.close()
```

# Trabalhando com Arquivos - Ler

```
>>> meu_arquivo = open("ola.txt","r")
>>> print(meu_arquivo.readlines())
['linha 1\n', 'linha 2\n', 'linha 3\n', 'Nova linha\n', 'Nova linha']
>>> meu_arquivo.close()
```

```
>>> meu_arquivo = open("ola.txt","r")
>>> for linha in meu_arquivo.readlines():
...     print(linha)
...
linha 1
linha 2
linha 3
Nova linha
Nova linha
>>> meu_arquivo.close()
```

# Trabalhando com Arquivos - Ler

```
>>> meu_arquivo = open("ola.txt","r")
>>> for linha in meu_arquivo.readlines():
...     print(linha,end="") # a saída é uma string vazia qdo não tem nada
...
linha 1
linha 2
linha 3
Nova linha
Nova linha>>>
>>> meu_arquivo.close()
```

# Trabalhando com Arquivos - Ler

```
>>> meu_arquivo = open("ola.txt","r")
>>> linha = meu_arquivo.readline() # lê a primeira linha
>>> while linha != "":
...     print(linha,end="")
...     linha = meu_arquivo.readline() # lê a próxima linha
...
linha 1
linha 2
linha 3
Nova linha
Nova linha>>>
>>>
>>> meu_arquivo.close()
```

# Trabalhando com Arquivos - Ler

```
>>> with open("ola.txt","r") as meu_arquivo:  
...     for linha in meu_arquivo.readlines():  
...         print(linha,end="")  
...  
linha 1  
linha 2  
linha 3  
Nova linha  
Nova linha>>>
```

# Trabalhando com Arquivos - Ler

```
>>> with open("ola.txt","r") as meu_arquivo, open('ola2.txt','w') as destino:  
...     for linha in meu_arquivo.readlines():  
...         destino.write(linha)  
...
```

# Trabalhando com Arquivos - Pesquisa .seek()

```
>>> meu_arquivo = open("ola.txt","r")
>>> print("primeira linha:",meu_arquivo.readline())
primeira linha: linha 1

>>> meu_arquivo.seek(0) # vai para o inicio
0
>>> print("primeira linha novamente:",meu_arquivo.readline())
primeira linha novamente: linha 1

>>> print("próxima linha:",meu_arquivo.readline())
próxima linha: linha 2

>>> meu_arquivo.seek(4) # vai para o caracter de indice 3
4
>>> print("linha 0 (4 caracter):",meu_arquivo.readline())
linha 0 (4 caracter): a 1
```

# O módulo OS

Para fazer algo mais avançado com estruturas de arquivo, você pode usar o módulo **os**, que expõe várias funções relacionadas ao sistema operacional. A primeira coisa que você precisará fazer é importar os para o seu código.

Se você está acostumado a trabalhar na linha de comando, o módulo os fornece muitas das mesmas funcionalidades básicas que provavelmente já serão familiares.

Por exemplo, a função `rmdir()` exclui um diretório e a função `mkdir()` cria um novo diretório:

# O módulo OS

Em breve, você verá como manipular e interagir com os arquivos de exemplo incluídos de várias maneiras. Embora existam muitas maneiras diferentes de configurar seus scripts corretamente para acessar arquivos, seguiremos um padrão simples neste curso.

Sempre que precisarmos escrever um script que utilize um arquivo de exemplo na pasta do curso, começaremos com algo como o seguinte código:

# O módulo OS

```
>>> import os  
>>> path = "/home/allan/Documentos/Projetos/Curso Pthon/exercicios"  
>>> os.mkdir(os.path.join(path,"meu_diretorio"))  
  
>>> import os  
>>> path = "/home/allan/Documentos/Projetos/Curso Pthon/exercicios"  
>>> os.rmdir(os.path.join(path,"meu_diretorio"))  
  
>>> import os  
>>> path = "/home/allan/Documentos/Projetos/Curso Pthon/exercicios"  
>>> for arquivo in os.listdir(path):  
...     print(arquivo)
```

# Arquivos CSV

```
>>> import csv
>>> import os
>>> path = "/home/allan/Documentos/Projetos/Curso Python/exercicios"
>>> with open(os.path.join(path,"dados.csv"), "r") as meu_arquivo:
...     reader = csv.reader(meu_arquivo)
...     for row in reader:
...         print(row)
...
```

# Arquivos CSV

```
>>> import os
>>> path = "/home/allan/Documentos/Projetos/Curso Python/exercicios"
>>> with open(os.path.join(path,"dados.csv"), "r") as meu_arquivo:
...     reader = csv.reader(meu_arquivo)
...     next(reader)
...     for nome,sobrenome,vendas in reader:
...         print(f"{nome} {sobrenome} vendeu: {vendas}")
...
```

# Arquivos CSV

```
>>> import csv
>>> import os
>>> path = "/home/allan/Documentos/Projetos/Curso Pthyon/exercicios"
>>> with open(os.path.join(path,"dados.csv"), "r") as meu_arquivo:
...     reader = csv.reader(meu_arquivo, delimiter="\t")
...     next(reader)
...     for row in reader:
...         print(row)
...
['Nome, Sobrenome, Vendas']
['Pedro, Silva, 3000']
['Maria, dos Santos, 5000']
['Ana, Maria, 8000']
```

# Arquivos CSV

```
>>> import csv
>>> import os
>>> path = "/home/allan/Documentos/Projetos/Curso Python/exercicios"
>>> with open(os.path.join(path,"dados2.csv"), "r") as meu_arquivo:
...     reader = csv.reader(meu_arquivo, delimiter=";")
...     next(reader)
...     for row in reader:
...         print(row)
...
```

# Arquivos CSV

```
>>> import os
>>> import csv
>>>
>>> path = "/home/allan/Documentos/Projetos/Curso Python/exercicios"
>>>
>>> classificacao = [["Filme","Tipo"],
...                     ["Uakim Dedi","Gente que não morre"],
...                     ["Pinóculos","Boneco com miopia"]]
>>> with open(os.path.join(path,"filmes1.csv"), "w") as meu_arquivo:
...     writer = csv.writer(meu_arquivo)
...     writer.writerow(classificacao)
```

# Pacotes de Terceiros

Provavelmente, o pip foi incluído quando você instalou o Python. Para determinar se você possui ou não o pip instalado em sua máquina, execute o seguinte em um terminal:

```
pip --version  
pip 19.2.3 from /home/allan/.local/lib/python3.7/site-packages/pip (python 3.7)
```

Package	Version
apturl	0.5.2
attrs	19.1.0

```
pip install requests
```

# Pacotes de Terceiros

Instale os pacotes:

- requests
- chardet
- certifi
- idna
- urllib3

```
pip3.7 show requests
Name: requests
Version: 2.22.0
Summary: Python HTTP for Humans.
Home-page: http://python-requests.org
Author: Kenneth Reitz
Author-email: me@kennethreitz.org
License: Apache 2.0
Location: /home/allan/.local/lib/python3.7/site-packages
Requires: certifi, chardet, idna, urllib3
Required-by: zenpy, webhooks, pipedrive-python-lib
```

# Pacotes de Terceiros

Para desinstalar um pacote utilize:

`pip uninstall <modulo>`

# Criando e Modificando Arquivos PDF

Atualmente, os arquivos PDF se tornaram uma espécie de mal necessário.

Apesar do uso frequente, é difícil trabalhar com PDFs.

Felizmente, o ecossistema Python possui ótimos pacotes para ler, manipular e criar arquivos PDF!

# Trabalhando com o conteúdo de um PDF

Nesta seção, você aprenderá como ler o conteúdo de um arquivo PDF usando o pacote PyPDF2. Antes de fazer isso, no entanto, você precisa instalar o PyPDF2 com o pip:

```
pip install pyPDF2 --user
```

```
$ pip3.7 install PyPDF2==1.26.0
```

# Trabalhando com o conteúdo de um PDF

```
>>> from PyPDF2 import PdfFileReader  
>>> path = "/home/allan/Documentos/Projetos/Curso Python/exercicios/Pride and Prejudice.pdf"  
>>> input_pdf = PdfFileReader(path)
```

```
>>> input_pdf.getNumPages()  
234
```

```
>>> input_pdf.getDocumentInfo()
```

# PDF - Extrair um Texto a partir de uma Página

```
>>> page0 = input_pdf.getPage(0)
>>> page0.extractText()
' \n \nThe Project Gutenberg EBook of Pride and Prejudice, by Jane Austen\n \n \nThis
eBook is for the use of anyone anywhere at no cost and with\n \nalmost no restrictio
ns whatsoever. You may copy it, give it away or\n \nre\n-\nuse it under the terms of
the Project Gutenberg License included\n \nwith this eBook or online at www.gutenber
g.org\n \n \n \nTitle: Pride and Prejudice\n \n \nAuthor: Jane Austen\n \n \nRelease
Date: August 26, 2008 [EBook #1342]\n \n [Last updated: August 11, 2011]\n \n \nLangua
ge: English\n \n \nCharacter set encoding: ASCII\n \n \n*** START OF THIS PROJECT G
UTENBERG EBOOK PRIDE AND PREJUDICE ***\n \n \n \n \n \n \nProduced by Anonymous Voluntee
rs, and David Widger\n \n \n \n \n \n \n \nPRIDE AND PREJUDICE \n \n \nBy Jane Austen
\n \n \n \n \n \nContents\n \n '
```

# PDF - Criar um PDF

```
$ pip3 install reportlab
```

```
>>> from reportlab.pdfgen import canvas  
>>> c = canvas.Canvas("hello.pdf")  
>>> c.drawString(100,100,"Olá Mundo")  
>>> c.save()
```

# Trabalhando com BD

Um banco de dados é um sistema estruturado para armazenar dados. Pode ser composto de vários arquivos CSV organizados em diretórios ou algo mais elaborado.

O Python vem com um banco de dados leve, chamado SQLite, perfeito para aprender a trabalhar com bancos de dados.

# Trabalhando com BD

```
>>> import sqlite3
>>> connection = sqlite3.connect("teste_data.db")
>>> cursor = connection.cursor() # Interação com o BD
>>> type(cursor)
<class 'sqlite3.Cursor'>

>>> query = "SELECT datetime('now','localtime');"
>>> cursor.execute(query)
<sqlite3.Cursor object at 0x7fa11c09ac70>

>>> cursor.fetchone()
('2019-10-14 16:16:17',)
```

# Trabalhando com BD

```
>>> time = cursor.execute(query).fetchone()[0]
>>> time
'2019-10-14 16:18:09'
```

```
>>> connection.close() # fecha a conexão
```

# Trabalhando com BD

```
>>> with sqlite3.connect("teste_data.db") as connection:  
...     cursor = connection.cursor()  
...     query = " SELECT datetim('now','localtime');"  
...     time = cursor.execute(query).fetchone()[0]  
...  
>>> time  
'2019-10-14 16:20:17'
```

# Trabalhando com BD

```
>>> import sqlite3
>>>
>>> connection = sqlite3.connect("teste_data.db")
>>> cursor = connection.cursor()
>>> cursor.execute(
...     """CREATE TABLE Pessoas(
...         Nome TEXT,
...         Sobrenome TEXT,
...         Idade
...     );"""
... )
<sqlite3.Cursor object at 0x7fa11c09ac70>
>>> cursor.execute(
...     """INSERT INTO Pessoas VALUES(
...         'Pedro',
...         'Silva',
...         42
...     );"""
... )
<sqlite3.Cursor object at 0x7fa11c09ac70>
>>> connection.commit()
>>> connection.close()
```

# Trabalhando com BD

```
>>> connection = sqlite3.connect("teste_data.db")
>>> cursor = connection.cursor()
>>> cursor.execute("SELECT * FROM Pessoas;")
<sqlite3.Cursor object at 0x7fa11c09a9d0>
>>> cursor.fetchone()
('Pedro', 'Silva', 42)
```

# Interagindo com a Web

```
>>> from urllib.request import urlopen  
>>> url = "http://htmeletronica.com.br/"  
>>> pagina = urlopen(url)  
>>> html = pagina.read().decode("utf-8")  
>>> print(html)
```

```
>>> resp = urlopen('https://github.com')  
>>> print(resp.read())
```

# Interagindo com a Web

```
>>> import ssl  
>>> ssl.OPENSSL_VERSION  
'OpenSSL 1.0.2g 1 Mar 2016'  
>>> urlopen('https://www.google.com')  
<http.client.HTTPResponse object at 0x7fa11c0e8210>  
>>> urlopen('https://www.google.com').read()
```

# Interagindo com a Web

\$ pip3 install MechanicalSoup

```
>>> import mechanize
>>> browser = mechanize.Browser()
>>> url = "http://olympus.realpython.org/login"
>>> page = browser.get(url)
>>> page.soup
<html>
<head>
<title>Log In</title>
</head>
<body bgcolor="yellow">
<center>
<br/><br/>
<h2>Please log in to access Mount Olympus:</h2>
<br/><br/>
<form action="/login" method="post" name="login">
Username: <input name="user" type="text"/><br/>
Password: <input name="pwd" type="password"/><br/><br/>
<input type="submit" value="Submit"/>
</form>
</center>
</body>
</html>
```

# Interagindo com a Web

```
$ pip3 install MechanicalSoup
```

# Interagindo com a Web

\$ pip3 install MechanicalSoup

```
import mechanize

# 1

browser = mechanize.Browser()
url = "http://olympus.realpython.org/login"
login_page = browser.get(url)
login_html = login_page.read()

# 2

form = login_html.select("form")[0]
form.select("input")[0]["value"] = "zeus"
form.select("input")[1]["value"] = "ThunderDude"

# 3

profiles_page = browser.submit(form, login_page.url)
```

# Interagindo com a Web

\$ pip3 install MechanicalSoup

```
>>> import mechanize
>>> browser = mechanize.Browser()
>>> url = "http://olympus.realpython.org/login"
>>> login_page = browser.get(url)
>>> login_html = login_page.read()
>>> form = login_html.select("form")[0]
>>> form.select("input")[0]["value"] = "zeus"
>>> form.select("input")[1]["value"] = "ThunderDude"
>>> profiles_page = browser.submit(form, login_page.url)
>>> profiles_page.url
'http://olympus.realpython.org/profiles'
>>> links = profiles_page.read().select("a")
>>> for link in links:
...     address = link["href"]
...     text = link.text
...     print(f"{text}:{address}")
...
Aphrodite:/profiles/aphrodite
Poseidon:/profiles/poseidon
Dionysus:/profiles/dionysus
```

# Interagindo com a Web

```
>>> import webbrowser
>>> webbrowser.register("xdg-open", None, webbrowser.BackgroundBrowser("xdg-open"), preferred=True)
>>> print(webbrowser.get()) # Now you will see <webbrowser.BackgroundBrowser object at 0x7f1e5373a048>
<webbrowser.BackgroundBrowser object at 0x7f6be6ad0650>
>>> webbrowser.open('https://google.com')
True
>>> Abrindo em uma sessão de navegador existente.
```

# Computação Científica e Gráfico - NumPy

```
$ pip3 install numpy
```

```
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
>>> matrix[0][1]  
2  
>>> for row in matrix:  
...     for i in range(len(row)):  
...         row[i] = row[i] * 2  
...  
>>> matrix  
[[2, 4, 6], [8, 10, 12], [14, 16, 18]]
```

# Computação Científica e Gráfico - NumPy

```
>>> import numpy as np
>>> matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> matrix
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> 2 * matrix
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
>>> matrix.shape
(3, 3)
>>> matrix.diagonal()
array([1, 5, 9])
>>> matrix.min()
1
>>> matrix.max()
9
```

# Computação Científica e Gráfico - Matplotlib

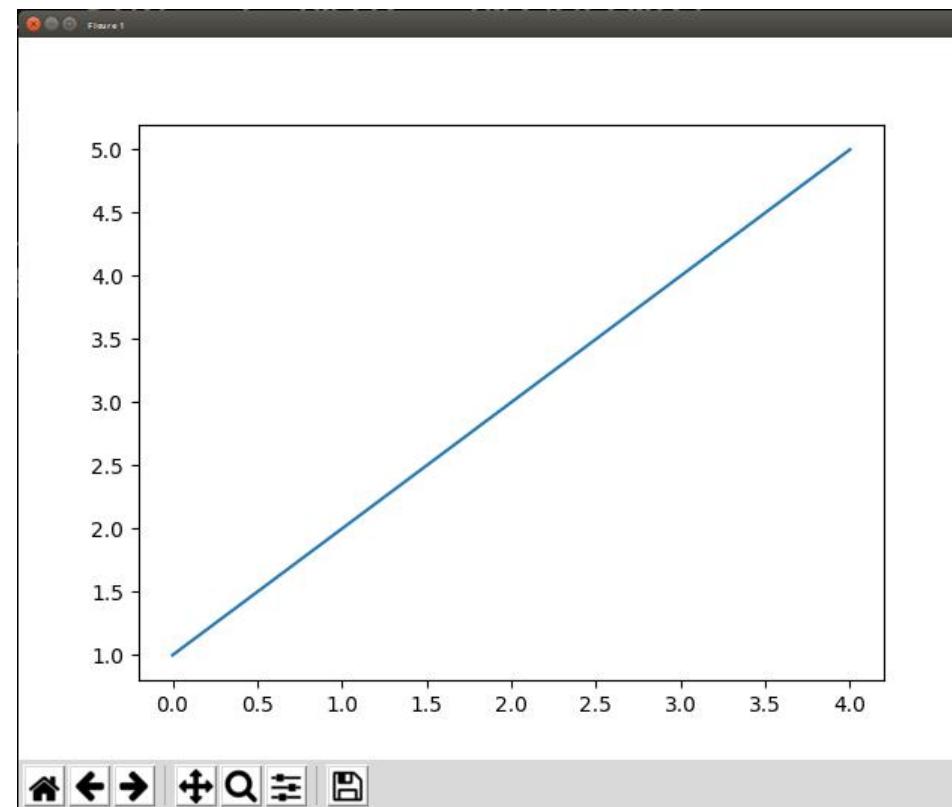
pip3 install matplotlib

pip3 show matplotlib

```
>>> from matplotlib import pyplot as plt  
>>> plt.plot([1, 2, 3, 4, 5])  
[<matplotlib.lines.Line2D object at 0x7f7f9f648c10>]  
>>> plt.show()
```

# Computação Científica e Gráfico - Matplotlib

```
>>> from matplotlib import pyplot as plt  
>>> plt.plot([1, 2, 3, 4, 5])  
[<matplotlib.lines.Line2D object at 0x7f7f9f648c10>]  
>>> plt.show()
```



# Computação Científica e Gráfico - Matplotlib

```
>>> from matplotlib import pyplot as plt  
>>> xs = [1, 2, 3, 4, 5]  
>>> ys = [2, 4, 6, 8, 10]  
>>> plt.plot(xs, ys)  
[<matplotlib.lines.Line2D object at 0x7f7f9f37bb10>]  
>>> plt.show()
```

```
>>> xs = [1, 2, 3, 4, 5]  
>>> ys = [3, -1, 4, 0, 6]  
>>> plt.plot(xs, ys)  
[<matplotlib.lines.Line2D object at 0x7f7f9f2af090>]  
>>> plt.show()
```

# Computação Científica e Gráfico - Matplotlib

```
from matplotlib import pyplot as plt
import numpy as np

days = np.arange(0, 21)
other_site = np.arange(0, 21)
real_python = other_site ** 2

plt.plot(days, other_site)
plt.plot(days, real_python)
plt.xticks([0, 5, 10, 15, 20])
plt.xlabel("Days of Reading")
plt.ylabel("Amount of Python Learned")
plt.title("Python Learned Reading Real Python vs Other Site")
plt.legend(["Other Site", "Real Python"])
plt.show()
```

# Computação Científica e Gráfico - Matplotlib

```
from matplotlib import pyplot as plt

xs = [1, 2, 3, 4, 5]
tops = [2, 4, 6, 8, 10]

plt.bar(xs, tops)
plt.show()
```

# Computação Científica e Gráfico - Matplotlib

```
from matplotlib import pyplot as plt
```

```
fruits = {  
    "apples": 10,  
    "oranges": 16,  
    "bananas": 9,  
    "pears": 4,  
}
```

```
plt.bar(fruits.keys(), fruits.values())  
plt.show()
```

# Computação Científica e Gráfico - Matplotlib

```
from matplotlib import pyplot as plt  
from numpy import random  
  
plt.hist(random.randn(10000), 20)  
plt.show()
```

# Computação Científica e Gráfico - Matplotlib

```
from matplotlib import pyplot as plt  
import numpy as np  
  
xs = np.arange(1, 6)  
tops = np.arange(2, 12, 2)  
  
plt.bar(xs, tops)  
plt.savefig("bar.png")
```

# GUI - easygui

```
$ pip3 install easygui
```

```
>>> import easygui as gui  
>>> gui.msgbox("Ola, Easy", "Msg Box", "Uauuu")
```

# GUI - Tkinter

```
>>> import tkinter as tk  
>>> window = tk.Tk()
```

# GUI - Tkinter

```
>>> import tkinter as tk  
>>> window = tk.Tk()  
>>> button = tk.Button()  
>>> button["text"] = "Click me!"  
>>> button["width"] = 25  
>>> button["height"] = 5  
>>> button["background"] = "blue"  
>>> button["foreground"] = "yellow"  
>>> button.pack()  
>>> window.mainloop()
```

# Enviando Email

```
import smtplib
from email.mime.multipart import MIME Multipart
from email.mime.text import MIMEText

fromaddr = "SEU EMAIL@GMAIL.COM"
toaddr = "EMAIL_DESTINO@PROVEDOR"
msg = MIME Multipart()
msg[ 'From' ] = fromaddr
msg[ 'To' ] = toaddr
msg[ 'Subject' ] = "Python Email"
body = "Enviando um Email pelo Google com Python"
msg.attach(MIMEText(body, 'plain'))

s = smtplib.SMTP('smtp.gmail.com', 587)
s.starttls()
s.login(fromaddr, "SENHA_DO_EMAIL_GOOGLE")
text = msg.as_string()
s.sendmail(fromaddr, toaddr, text)
s.quit()
```

# Enviando Email

```
import smtplib
from email.mime.multipart import MIME Multipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders

email_user = 'EMAIL_ORIGEM'
email_send = 'EMAIL_DESTINO'
subject = 'Python'

msg = MIME Multipart()
msg[ 'From' ] = email_user
msg[ 'To' ] = email_send
msg[ 'Subject' ] = subject
```

# Enviando Email

```
body = "Estou enviando um arquivo anexo"
msg.attach(MIMEText(body, 'plain'))
filename = 'arquivo.xlsx'
attachment = open(filename, 'rb')
part = MIMEBase('application', "octet-stream")
part.set_payload(attachment.read())
encoders.encode_base64(part)
part.add_header('Content-Disposition', 'attachment; filename="arquivo.xlsx"')
```

# Enviando Email

```
msg.attach(part)
text = msg.as_string()
server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()
server.login(email_user, 'SUA_SENHA')
server.sendmail(email_user, email_send, text)
server.quit()
```

# Trabalhando com Excel

```
import xlsxwriter
# Criar um workbook and adicionar um worksheet.
workbook = xlsxwriter.Workbook('Despesas01.xlsx')
worksheet = workbook.add_worksheet()
# Alguns dados
expenses = (
    ['Aluguel', 1000], ['Gas', 100], ['iFood', 300], ['Smartfit', 50],
)

# Iniciando na Linha 0 e Col 0
row = 0
col = 0
# Iterando sobre os daddos e escrevendo a saida linha por linha.
for item, cost in (expenses):
    worksheet.write(row, col, item)
    worksheet.write(row, col + 1, cost)
    row += 1
# Escrever o total com uma formula
worksheet.write(row, 0, 'Total')
worksheet.write(row, 1, '=SUM(B1:B4)')
workbook.close()
```

# Trabalhando com Word

```
pip install image --user
```

```
pip install pillow --user
```

```
pip install pillow --upgrade --user
```

```
pip install python-docx --user
```

# Trabalhando com Word

```
from docx import Document
from docx.shared import Inches

document = Document()
document.add_heading('Document Title', 0)

p = document.add_paragraph('A plain paragraph having some ')
p.add_run('bold').bold = True
p.add_run(' and some ')
p.add_run('italic. ').italic = True
```

# Trabalhando com Word

```
document.add_heading('Heading, level 1', level=1)
document.add_paragraph('Intense quote', style='Intense Quote')

document.add_paragraph(
    'first item in unordered list', style='List Bullet'
)
```

# Trabalhando com Word

```
document.add_paragraph(  
    'first item in ordered list', style='List Number'  
)
```

# Trabalhando com Word

```
#document.add_picture('C:\\Users\\soumi\\Pictures\\application.jpg', width=Inches(1.25))
recordset = [
    {
        "id" : 1,
        "qty": 2,
        "desc": "New item"
    },
    {
        "id" : 2,
        "qty": 2,
        "desc": "New item"
    },
    {
        "id" : 3,
        "qty": 2,
        "desc": "New item"
    }
]
```

# Trabalhando com Word

```
print(recordset[0]['id'])
table = document.add_table(rows=1, cols=3)
hdr_cells = table.rows[0].cells
hdr_cells[0].text = 'Id'
hdr_cells[1].text = 'Quantity'
hdr_cells[2].text = 'Description'
for item in recordset:
    #print(item)
    row_cells = table.add_row().cells
    row_cells[0].text = str(item['id'])
    row_cells[1].text = str(item['qty'])
    row_cells[2].text = str(item['desc'])

document.add_page_break()
document.save('simple.docx')
```

# Hora da Diversão

pip install arcade

```
import arcade

# tela
SCREEN_WIDTH = 600
SCREEN_HEIGHT = 600

# Abrir uma Janela
arcade.open_window(SCREEN_WIDTH, SCREEN_HEIGHT, "Exemplo de Desenho")
#cor de fundo
arcade.set_background_color(arcade.color.WHITE)
# Renderizar
arcade.start_render()
```

# Hora da Diversão

```
# Desenha uma face
x = 300
y = 300
radius = 200
arcade.draw_circle_filled(x, y, radius, arcade.color.YELLOW)

# Desenhar um olho direito
x = 370
y = 350
radius = 20
arcade.draw_circle_filled(x, y, radius, arcade.color.BLACK)
```

# Hora da Diversão

```
# Desenhar Olho esquerdo
x = 230
y = 350
radius = 20
arcade.draw_circle_filled(x, y, radius, arcade.color.BLACK)

# Sorriso
x = 300
y = 280
width = 120
height = 100
start_angle = 190
end_angle = 350
arcade.draw_arc_outline(x, y, width, height, arcade.color.BLACK, start_angle, end_angle, 10)
```

# Hora da Diversão

```
# Finalizar
arcade.finish_render()

# Executar
arcade.run()
```