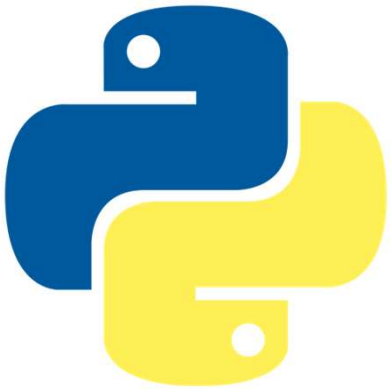


# Introdução a Linguagem Python

# Introdução

- Surgiu em 1989;
- Criada por Guido van Rossum;
- Nome inspirado em Monty Python and the Flying Circus;
- Licença compatível com Software Livre;
- Linguagem de altíssimo nível;
- Tipagem Dinâmica;
- Multiparadigma (OO, funcional e procedural);
- Compilada + Interpretada;
- Visa aumentar a produtividade do programador;




Grupo humorístico britânico Monty Python

# Filosofia do Python

- Bonito é melhor que feio
- Explícito é melhor do que implícito
- Simples é melhor do que complexo
- Complexo é melhor do que complicado
- Legibilidade conta

Zen of Python completo

Digite no python o comando abaixo para lê-lo:

```
import this
```


A Imagem acima não tem nenhuma relação com o texto, mas não podia deixar passar a oportunidade de fazer mais uma piada com Monty Python

# Compilar e Interpretar

- O código fonte é traduzido pelo Python para bytecode, que é um formato binário com instruções para o interpretador. O bytecode é multiplataforma e pode ser distribuído e executado sem fonte original.
- Por padrão, o interpretador compila o código e armazena o bytecode em disco, para que a próxima vez que o executar, não precise compilar novamente o programa, reduzindo o tempo de carga na execução.

# Compilar e Interpretar

- Quando um programa ou um módulo é evocado, o interpretador realiza a análise do código, converte para símbolos, compila (se não houver bytecode atualizado em disco) e executa na máquina virtual Python.
- O bytecode é armazenado em arquivos com extensão “.pyc” (bytecode normal) ou “.pyo” (bytecode otimizado). O bytecode também pode ser empacotado junto com o interpretador em um executável, para facilitar a distribuição da aplicação, eliminando a necessidade de instalar Python em cada computador.

# Compilar e Interpretar

- Modo Interativo:
  - O interpretador Python pode ser usado de forma interativa, na qual as linhas de código são digitadas em um terminal. Para evocar o modo interativo basta executar o interpretador:
  - O modo interativo é uma característica diferencial da linguagem, pois é possível testar e modificar trechos de código antes da inclusão do código em programas, fazer extração e conversão de dados ou mesmo analisar o estado dos objetos que estão em memória, entre outras possibilidades.

# Indentação

- Python foi desenvolvido para ser uma linguagem de fácil leitura, com um visual agradável, frequentemente usando palavras e não pontuações como em outras linguagens.
- Para a separação de blocos de código, a linguagem usa espaços em branco e indentação ao invés de delimitadores visuais como chaves (C, Java) ou palavras (BASIC, Fortran,Pascal).
- Diferente de linguagens com delimitadores visuais de blocos, em Python a indentação é obrigatória. O aumento da indentação indica o início de um novo bloco, que termina da diminuição da indentação.

# Comentários

- O caractere # marca o inicio de comentário. Qualquer texto depois do # será ignorado até o fim da linha , com exceção dos comentários funcionais.
- Para comentário em bloco, usa-se três aspas simples ao início e ao fim do bloco.

# Comentário Funcional

- É possível usar codificação diferente de ASCII() em arquivos de código Python. A melhor maneira de fazê-lo é através de um comentário adicional logo após a linha #!:

  *# -*- coding: encoding -*-*

- Definir o interpretador que será utilizado para rodar o programa em sistemas UNIX, através de um comentário começando com "#!" no inicio do arquivo, que indica o caminho para o interpretador (geralmente a linha de comentário será algo como "#!/usr/bin/envpython").

# Lista de Palavras Reservadas

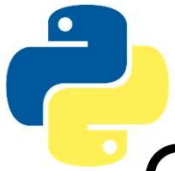
- and
- as
- assert
- break
- class
- continue
- def
- del
- elif
- else
- except
- exec
- finally
- for
- from
- global
- if
- import
- in
- is
- lambda
- not
- or
- pass
- print
- raise
- return
- try
- while
- with
- yield

# Operadores Matemáticos

| | |
|---|---|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão |
| % | Módulo |
| ** | Potenciação |
| // | Divisão por Inteiro |

# Operadores Relacionais

| | |
|---|---|
| == | Igual à |
| != | Diferente de |
| < | Menor que |
| <= | Menor ou Igual à |
| > | Maior que |
| >= | Maior ou Igual à |
| is | Identidade do Objeto |
| is not | não Identidade de Objeto |



# Operadores Lógicos

| | |
|---|---|
| and | E |
| or | OU |

Outra escrita

| | |
|---|---|
| & | E |
| \| | OU |

# Variáveis

Variáveis controlam dados da memória e possuem um nome e um tipo.

Vale lembrar que a tipagem dinâmica do Python identifica automaticamente o tipo de dados baseado no conteúdo atribuído.

Caso seja atribuído um conteúdo diferente a uma mesma variável ela automaticamente muda seu tipo.

# Variáveis

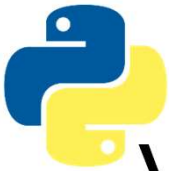
As variáveis são criadas automaticamente assim que um valor é atribuído a elas.

A Sintaxe para atribuição é:

*variável = valor*

Exemplo:

```
x=15
nom="José Silva"
```



# Variáveis

É possível remover uma variável com o comando del

Exemplo:

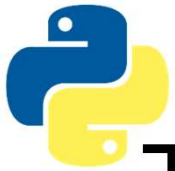
```
del nom
```

# Atribuição de dados

A atribuição de dados no Python é feito com o operador de atribuição =

Exemplo:

```
x =15
y=3.72
nom="José Silva"
lista_1=["Lucas Evangelista da Silva", 19, "Maria do Rosario", 83]
```



# Tipos de dados

| | |
|---|---|
| str | string, ou cadeia de caracteres imutáveis [1] |
| unicode | cadeia de caracteres imutáveis unicode |
| int | número inteiro |
| float | número com ponto flutuante |
| complex | número complexo |
| bool | booleano |
| list | lista heterogênea mutável |
| tuple | tupla imutável |
| dict | dicionário, ou conjunto associativo |
| set | conjunto não ordenado |
| frozenset | conjunto não ordenado, sem elementos duplicados |

# Tipos de dados

É possível checar o tipo de dados de uma variável dando a instrução type(variável)

Exemplo:

```
x=15
type(x)
nom="José Silva"
type(nom)
```

Se for no modo compilado não esqueça do print()

```
x=15
print(type(x))
```

# Entrada de dados no terminal

A entrada de dados no modo Interativo ou Script em terminal se dá pelo comando input

Na representação, variável recebe valor da entrada

Exemplo:

```
x=input()
```

Para adicionar uma mensagem na entrada, basta escrever um texto dentro do parênteses:

```
x=input("Digite um Nome")
```

# Entrada de dados no terminal

A entrada de dados normal vai adicionar o conteúdo sempre na forma de string, mesmo se digitado explicitamente um número.

Para indicarmos que a entrada é de um tipo diferente temos que fazer a conversão dela antes de atribuir o valor a variável.

Exemplo:

```
w=int(input("Digite um Número"))
r=float(input("Digite um Valor"))
h=bool(input("Digite True ou False"))
```

# Saída de dados no terminal

A saída de dados no modo Interativo ou Script em terminal se dá pela função print()

Exemplo:

```
print("Existe")
print(w)
```

Para intercalar valores e variáveis separe-os por virgula:

```
print(w, “ Existe")
```

# Saída de dados no terminal

É possível usar estilos de formatação variados no python veja os exemplos:

```
nom=“José”

#estilo de formatação problematico
print(“Bom dia“ + nom + “!”)

#estilo de formatação tipo C
print(“Bom dia %s !“ % nom )

#estilo de formatação novo
print(“Bom dia {} !” .format(nom))
```

# Saída de dados no terminal

Outro exemplo de formatação:

```
>>> print('{0} {1} {0} {0} {1}' . format("tchu", "tcha"))

tchu tcha tchu tchu tcha
```


Desculpe, por isso!

# Saída de dados no terminal

Outro exemplo de formatação:

```
tabela = '{dsemana:40} {prato:10}'
print(tabela.format(dsemana='Segunda', prato='Ovo'))
print(tabela.format(dsemana='Terça', prato='Carne Moída'))
```

# Blocos e Indentação

Todos os conteúdos das estruturas de bloco de controle (if, while, for, funções ...) são considerados blocos

O controle do bloco é feito por indentação que consiste em um recuo de 4 espaços.

Tudo que pertencer a aquele bloco específico ficará com o recuo de 4 espaços.

OBS: Os programas de edição de códigos voltados para Python já consideram a tabulação (tecla Tab) como 4 espaços, mas sempre é bom conferir.

# Blocos e Indentação

A linha anterior ao inicio do bloco é demarcada por dois pontos :

Exemplos:

```
if x==1:
    print("Existe")
else:
    print("Não existe")
#-----
while True:
    print(“está valendo”)
#-----
funcao ():
    x=x+1
    return x
```

# Decisão

A estrutura de decisão mais básica em Python é a estrutura if (se, em português). E Sempre conterá um teste relacional/lógico.

Sua sintaxe é:

```
if teste lógico:
    comandos executados caso verdadeiro
```

# Decisão

Exemplos Simples:

```
if qt==0:
    print(“compre mais bolachas”)
```

```
if qt==0:
    print(“compre mais bolachas”)
else:
    print(“não acabou ainda”)
```

# Decisão

Exemplos com operadores lógicos:

```
if qt>=0 and qt<=20 :
    print(“O Numero está entre a faixa de zero e vinte”)
else:
    print(“O Numero está fora da faixa”)
```

```
if qt<0 or qt>20 :
    print (“O Numero está fora da faixa”)
else:
    print (“O Numero está entre a faixa de zero e vinte”)
```

# Decisão

Exemplo de decisão aninhada com elif:

```
if qt==0:
    print(“é neutro”)
elif qt>0 :
    print(“é positivo”)
else:
    print(“só sobrou o negativo”)
```

# Decisão

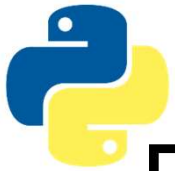
Exemplo de decisão aninhada com elif:

```
if op==0:
    print(“Encerrar ligação”)
elif op==1:
    print(“Falar sobre sua conta”)
elif op==2:
    print(“Falar com Suporte”)
elif op==3 :
    print(“Falar com o Zé do Caixão”)
elif op==4:
    print(“Cancelar sua Assinatura”)
elif op==5:
    print(“Falar com um de nossos atendentes”)
else:
    print(“Desculpe opção não cadastrada”)
```

# Repetições

As estruturas de repetição mais usadas em Python são o **while** e o **for**.

Sendo o **for** uma estrutura muito versátil para se utilizar em conjunto com variáveis compostas como veremos adiante



# Repetições

Estrutura **while**.

```
while <teste lógico> :
    <instruções a serem executadas enquanto teste lógico for verdadeiro>
```

# Repetições

## Repetição infinita

```
while True:
    print("Isto é uma repetição infinita")
```

O break interrompe a repetição a qualquer momento, mesmo se o teste lógico ainda for verdadeiro

```
while True:
    valor=(int(input("Isso será uma repetição infinita a menos que digite 7")
    if valor==7:
        break
```

# Repetições

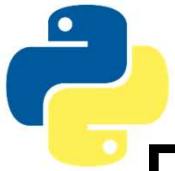
O while pode ser usado para uma quantidade determinada de vezes, porém você tem de fazer este controle.

```
x=0
while x<20:
    print(“O valor agora é: “,x)
    x=x+1
print(“\n\nAqui acabamos a repetição\n\n”)
```

# Repetições

Mas o while é também usado para números indeterminados de vezes.

```
x=9
while x!=0:
    x=int(input(“Digite um valor inteiro, ou zero para sair”)
    s=s+x
print(“\n\nAqui acabamos a repetição\n\nO valor da soma é: ”, s)
```



# Repetições

Estrutura **for** para um número determinado de vezes.

```
for <variável_de_controle> in rage (quantidade):
    <instruções a serem executadas enquanto teste lógico for verdadeiro>
```

# Repetições

Exemplo **for** para um número determinado de vezes.

```
for w in rage (8):
    print (w)
```

```
for z in rage (3, 11, 2):
    print (w)
```

```
for q in rage (11, 3, -2):
    print (q)
```

# Repetições

O **for** também pode ser usado para “varrer” os itens dentro de uma lista ou tupla:

```
lista=[1.4, 33, “casa”, 7.2, 99, “Outros”]
for t in lista :
    print (t)
```

Ou de uma string:

```
for u in (“Boa Noite SP”):
    print (u)
```

# Repetições

O **for** aninhado :

```
for l in range (3) :
    for k in range (10) :
        print (l, k)
    print("----------")
```