import this

Uma breve introdução à linguagem python

CC BY SA

python

import this

Uma breve introdução à linguagem python

$whoami

python

import this

Uma breve introdução à linguagem python

A Linguagem Python foi concebida no fim dos anos 80. Criado por Guido Van Rossum, teve seu primeiro contato com o mundo externo no dia 20 de fevereiro de 1991, quando foi publicado no no grupo de notícias 'alt.sources'. Essa versão foi numerada como 0.9.0 e liberada sob uma licença que era quase uma cópia exata da licença MIT usada pelo projeto X11 na época.

> *"Então, como quase tudo que eu escrevi, Python era de código aberto (N.T.: open source, no original) antes que o termo fosse inventado por Eric Raymond e Bruce Perens ao final de 1997."*

- Guido van Rossum

import this

Uma breve introdução à linguagem python

O nome da linguagem não faz referência ao réptil, a inspiração veio de um dos grupos de comédia favoritos de Guido van Rossum:

Monty Python's Flying Circus

python

import this

Uma breve introdução à linguagem python

“Espera um minuto, e essa cobra no logo aí embaixo?”

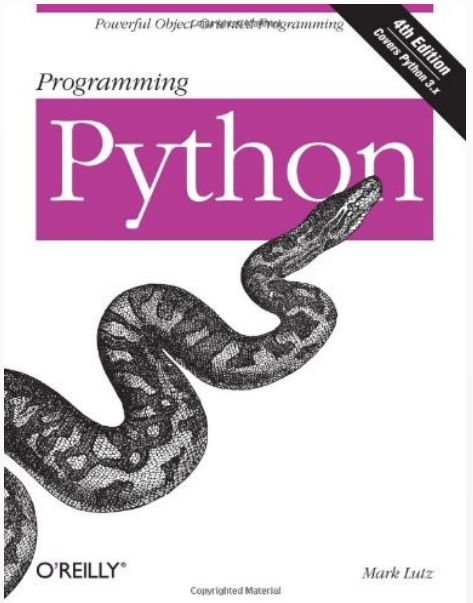
python

import this

Uma breve introdução à linguagem python

Por muitos anos Guido van Rossum resistiu à tentação de associar a linguagem com cobras. Finalmente desistiu quando O'Reilly queria colocar uma cobra na capa de seu primeiro livro sobre Python "Programming Python".

> *"Era tradição da O'Reilly usar imagens de animais e, se deveria ser um animal, que fosse uma cobra."*
>
> - Guido van Rossum



python

Coisas legais do Python

- produtividade e legibilidade

- sintaxe simples e clara, mas muito expressiva

- o uso de identação para marcar blocos

- não é preciso usar ponto e vírgula para separar as instruções \o/

- suporta múltiplos paradigmas de programação

- curva de aprendizado que permite fazer muita coisa em pouco tempo

- vem com baterias inclusas

- mas se precisar, tem pacotes (bibliotecas) para todo tipo de coisa

- livre e multiplataforma

OK! Tem muita coisa legal, mas quem usa python?

python

import this

Uma breve introdução à linguagem python

python

import this

Uma breve introdução à linguagem python

Quem usa?

http://wiki.python.org/moin/OrganizationsUsingPython

python

import this

Uma breve introdução à linguagem python

python

Vamos aquecer os dedos!!

python

O Interpretador Interativo Python

Abra o terminal e digite:

python3

Agora você está no ambiente do interpretador interativo do python.

Para quem nunca escreveu um programa, vamos fazer um pequeno ritual de iniciação, digite:

>>>print(“Hello world!”)

O Interpretador Interativo Python

Agora digite o título ali de cima:

>>>import this

import this

Uma breve introdução à linguagem python

O Zen do Python, por Tim Peters

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.
Esparso 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.
Diante da ambiguidade, 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 frequentemente seja melhor que *já*.
Se a implementação é difícil de explicar, é uma má ideia.
Se a implementação é fácil de explicar, pode ser uma boa ideia.
Namespaces são uma grande ideia — vamos ter mais dessas!

Declarando uma variável

```
>>>texto = “Python possui tipagem dinâmica e forte”
>>>num = 2016
>>>pi = 3.14159265
```

O sinal de igualdade (=) indica uma atribuição, portanto é atribuído à variável **texto** o valor *“Python possui tipagem dinâmica e forte”*. O mesmo para as variáveis **num**, que recebe *2016* e **pi**, que recebe *3.14159265*

Declarando uma variável

Python não te obriga a declarar o tipo das variáveis antes de atribuir algum valor a elas, mas é necessário inicializá-las.

```
>>> a = 10
>>> c = a + b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
```

Declarando uma variável

Tipagem dinâmica forte: são necessárias conversões explícitas.

```
>>> a = “10”
>>> b = 5
>>> c = a + b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Tipos de objetos embutidos do Python

Inteiro e Ponto Flutuante

O objeto do tipo **int** representa um número inteiro de base 10 e **float** representa um número real. O objeto **int** permite operações matemáticas.

Tipos de objetos embutidos do Python

```
>>> 2 + 3          #Soma
5
>>> 7 – 4          #Subtração
3
>>> 6 * 2          #Multiplicação
12
>>> 6 ** 2         #Potenciação
36
>>> 7 / 2          #Divisão
3.5
>>> 7 // 2         #Divisão devolvendo somente a parte inteira
3
>>> 7%2            #Divisão devolvendo o resto
1
```

Tipos de objetos embutidos do Python

Também é possível importar pacotes com ferramentas adicionais para lidar com esses dois objetos.

```
>>> import math

>>> math.pi
3.141592653589793

>>> math.sqrt(16)
4.0
```

Tipos de objetos embutidos do Python

“String”

O objeto do tipo **str** é uma cadeia de caracteres, definida sempre entre aspas simples ou duplas. Strings são usadas para gravar tanto uma informação de texto (o seu nome, por exemplo), quanto um coleção arbitrária de bytes (como o conteúdo de um arquivo de imagem).
Uma string também pode ser criada utilizando a função *str()*

```
>>> str(42)
'42'
>>> var = 13
>>> var = str(var)
>>> var
'13'
```

Tipos de objetos embutidos do Python

```
>>>var = “Python”
```

P	y	t	h	o	n
0	1	2	3	4	5

```
>>>var[0]
'P'
>>>var[5]
n

#Espaços também contam como caracteres
```

Tipos de objetos embutidos do Python

Strings possuem alguns métodos, tais como:

```
>>>var.replace(P, p)
'python'
```

Porém, *.replace* não altera **var**, apenas retorna um resultado

```
>>>var
'Python'
```

Para armazenar essa alteração é necessário atribuir o valor retornado a alguma variável ou a própria variável **var**

```
>>>var = var.replace('P', 'p')
>>>var
'python'
```

Tipos de objetos embutidos do Python

.upper() - Retorna uma cópia da string com todos os caracteres convertidos para caixa alta.

.split(sep=None, maxsplit=-1) – Retorna uma lista das palavras na string, usando *sep* como um delimitador para a string e *maxstring* para limitar o número de recortes.

.swapcase() - Retorna uma cópia da string com os caracteres de caixa alta em caixa baixa e vice-versa.

.title() - Retorna uma cópia da string com a primeira letra das palavras em caixa alta.

Outros métodos:
https://docs.python.org/3.5/library/stdtypes.html#str

Tipos de objetos embutidos do Python

É possível concatenar strings com o operador **+**

```
>>> var = 'python é uma linguagem de programação'
>>> var.title()
'Python É Uma Linguagem De Programação'

>>> titulo1 = var[0:6].title()
>>> titulo1
'Python'

>>> titulo2 = var[6:]
>>> titulo2
' é uma linguagem de programação'
>>> titulo1 + titulo2
'Python é uma linguagem de programação'
```

 # Para você ler e testar em outro momento:

Strings são imutáveis:

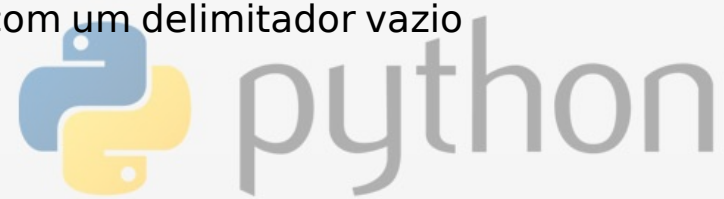
```
>>> S = 'Spam'
>>> S[0] = 'z'
# Immutable objects cannot be changed
...error text omitted...
TypeError: 'str' object does not support item assignment
```

No entanto é possível adicionar um caractere criando uma nova variável:

```
>>> S = 'z' + S[1:]
>>> S
'zpam'
```

Também é possível alterar a string expandindo ela em uma lista, passando quantos caracteres desejar, e juntando tudo em uma nova string

```
>>> S = 'shrubbery'
>>> L = list(S)                    # Expande para uma lista: [...]
>>> L
['s', 'h', 'r', 'u', 'b', 'b', 'e', 'r', 'y']
>>> L[1] = 'c'                     # Passa um novo caractere para a posição 1
>>> ''.join(L)                     # Junta em uma string com um delimitador vazio
'scrubbery'
```



Tipos de dados

[“Listas”]

Delimitadas por [ ], são uma coleção mutável e sem tamanho fixo. Listas aceitam vários tipos de objetos:

```
>>>lista = [ 1, 3.14, 'Python', [42, 'Linux'], ('Tuplas, 2'), {'dict': 1} ]
```

Algumas operações aceitas pelas listas:

```
>>> lista[0]
1
>>> lista[:3]
[1, 3.14, 'Python']
>>> lista[::2]
[1, 'Python', 'Tuplas']
```

Tipos de dados

```
>>> lista[3] = 'Debian'
>>> lista
[1, 3.14, 'Python', 'Debian', ('Tuplas', 2), {'dict': 1}]

>>> lista + [3,4,5]
[1, 3.14, 'Python', 'Debian', ('Tuplas', 2), {'dict': 1}, 3, 4, 5]

>>> lista * 2
[1, 3.14, 'Python', 'Debian', ('Tuplas', 2), {'dict': 1}, 1, 3.14, 'Python',
'Debian', 'Tuplas', {'dict': 1}]

>>> lista = [1, 3.14, 'Python', 'Debian', ('Tuplas', 2), {'dict': 1}]
>>> lista.append('software livre')
>>> lista
[1, 3.14, 'Python', 'Debian', ('Tuplas', 2), {'dict': 1}, 'software livre']
```

Tipos de dados

```
>>> lista.pop()
'software livre'
>>> lista
[1, 3.14, 'Python', 'Debian', ('Tuplas', 2), {'dict': 1}]

>>> lista.pop(4)
('Tuplas', 2)
>>> lista
[1, 3.14, 'Python', 'Debian', {'dict': 1}]
```

Tipos de dados

```
>>> lista.reverse()
>>> lista
[{'dict': 1}, 'Debian', 'Python', 3.14, 1]

>>> lista_vogais = ['e', 'u', 'i','a', 'o']
>>> lista_vogais.sort()
>>> lista_vogais
['a', 'e', 'i', 'o', 'u']

>>> lista_vogais.sort(reverse=True)
>>> lista_vogais
['u', 'o', 'i', 'e', 'a']

>>> len(lista_vogais)
5
```

Tipos de dados

```
lista = [1, 3.14, 'Python', [42, 'Linux'], ('Tuplas', 2), {'dict': 1}]
>>> lista[3][1]
'Linux'
>>> lista[2][1]
'y'
```

Podemos representar matrizes utilizando as listas:

```
>>> matriz = [[1, 2, 3],
...           [4, 5, 6],
...           [7, 8, 9]]
>>> matriz[0]
[1, 2, 3]
>>> matriz[0][1]
2
>>> matriz[2][2]
9
```

Tipos de dados

Listas também podem ser criadas utilizando a função *list()*

```
>>> var = 1,2,3,'casa', {'dir':1}
>>> var
(1, 2, 3, 'casa', {'dir': 1})

>>> list(var)
[1, 2, 3, 'casa', {'dir': 1}]

>>> len(var)
5

>>> var = 'Python'
>>> list(var)
['P', 'y', 't', 'h', 'o', 'n']
```

Tipos de dados

(“Tuplas”)

Delimitadas por ( ), são uma coleção imutável e sem tamanho fixo. Geralmente usadas para representar uma coleção fixa, como um calendário, por exemplo. Também suportam vários tipos de objetos e operações de sequência:

```
>>> tupla = (1 ,3.14, 'Linux', ['Debian', 8], {'dict':1}, ('software', 'livre'))

>>> len(tupla)
6

>>> tupla + (5, 6, 7)
(1, 3.14, 'Linux', ['Debian', 8], {'dict': 1}, ('software', 'livre'), 5, 6, 7)
```

Tipos de dados

```
>>> tupla[2]
'Linux'

>>> tupla[2][0]
'L'

>>> tupla[5][1]
'livre'

>>> tupla[2] = 'Windows'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError:  Chega de tela azul!!
```

Tipos de dados

```
>>> tupla[3][0] = Python'
>>> tupla
(1, 3.14, 'Linux', ['Python', 8], {'dict': 1}, ('software', 'livre'))

>>> tupla2 = ('novo',) + tupla[1:]
>>> tupla2
('novo', 3.14, 'Linux', ['Python', 42], {'dict': 1}, ('software', 'livre'))
```

Tipos de dados

{“Dicionários”: “dict”}

Com a sintaxe { chave : valor }, um dicionário em Python é uma estruturas de dados mutável e não ordenada. Uma *chave* pode ser qualquer objeto não mutável, já o *valor* poder ser qualquer objeto visto até aqui.

```
>>> dic = { 'linux' : 'debian' }

>>> dic2 = { 'linux' :  'debian', 1 : 'python', (1,2,3) : 4}

>>> dic3 = { 'unisal' : { 'flisol' : [ 'linux', 'python', 2016 ] } }

>>> dic4 = dict( linux = 'Debian', flisol = 2016)
>>> dic4
{'linux': 'Debian', 'flisol': 2016}
```

Tipos de dados

```
>>> dic = {'linux':'debian', 'programação':'python', 'flisol': 2016}
>>> len(dic)
3

>>> dic['flisol']
2016

>>> dic['flisol'] = 'unisal'
>>> dic
{'linux': 'debian', 'flisol': 'unisal', 'programação': 'python'}

>>> del dic['flisol']
>>> dic
{'linux': 'debian', 'programação': 'python'}
```

Tipos de dados

```
>>> dic.pop('linux')
'debian'
>>> dic
{'flisol': 2016, 'programação': 'python'}

>>> dic.get('linux')
'debian'

>>> dic2 = { 'windows' : 'tela azul'}
>>> dic.update(dic2)
>>> dic
{'linux': 'debian', 'flisol': 2016, 'programação': 'python', 'windows': 'tela
azul'}
```

 # Para ler e testar em outro momento:

O que é mutável e imutável

Mutável: objetos que podem ter seus valores alterados mantendo o mesmo *id.*

```
>>> lista = [1,2,3,4,5]
>>> id(lista)
3073195948
>>> lista.append(9)
>>> lista
[1, 2, 3, 4, 9]
>>> id(lista)
3073195948
```

Imutável: objetos que não podem ter seus valores alterados. Para alterar o seu valor, um novo objeto deve ser criado, assim um novo *id* é atribuído. Tuplas, strings e inteiros são exemplos de objetos imutáveis.

print()

A função *print()* imprime na tela os valores dos argumentos passados para ela:

```
>>> print("Flisol 2016")
Flisol 2016

>>> arg = "Python"
>>> print(arg)
Python

>>> evento = "Flisol"
>>> ling = "Python"
>>> ano = 2016
>>> print("%s %d : Mini-curso de %s" %(evento, ano, ling))
Flisol 2016 : Mini-curso de Python
```

print()

%d e %i para inteiros

%f para float

%.2f específica quantas casas depois da vírgula

%s para strings

```
>>> num = 3.1415926535
>>> print("Pi aproximandamente igual %.4f" % num)
Pi aproximandamente igual 3.1416
```

print()

```
>>> var1 = 'Linux'
>>> var2 = 'Debian'
>>> var3 = 'Python'

>>> print(var1, var2, var3)
Linux Debian Python

>>> print(var1, var2, var3, sep=', ')
Linux, Debian, Python

>>> print(var1, var2, var3, sep=' - ')
Linux - Debian - Python

>>> print(var1, var2, var3, sep=', ', end='!!!\n')
Linux, Debian, Python!!!
```

input()

A função input() permite armazenar uma entrada do usuário feita pelo teclado.
Por padrão, o valor armazenado é um objeto do tipo *string.*

```
>>> nome = input("Nome completo: ")
Nome completo: Guido Van Rossum
>>> nome
'Guido Van Rossum'
```

Entretanto é possível converter uma entrada para inteiro, com a função *int( ).*

```
>>> idade = int(input("Idade: "))
Idade: 60
>>> idade
60
```

input()

Também é possível usar a função *list( ),* gerando uma lista com cada caractere como um elemento da lista.

```
>>> lista = list(input("Digite qualquer coisa: "))
Digite qualquer coisa: qualquer coisa
>>> lista
['q', 'u', 'a', 'l', 'q', 'u', 'e', 'r', ' ', 'c', 'o', 'i', 's', 'a']
```

Executando uma aplicação pelo terminal

Vamos abrir um editor de texto. A distribuição Debian já traz o gedit no pacote de programas, que vai servir bem aos nossos propósitos.
Digite *gedit&* no terminal, o símbolo *&* diz ao sistema para executar uma aplicação (no nosso caso o *gedit*) em segundo plano, deixando o terminal livre para outras operações.

Executando uma aplicação pelo terminal

Salve o arquivo com a extensão *.py*
Ex: *nomeDoPrograma.py*

Agora vamos escrever o seguinte programa:

```
nome = input("Digite seu nome: ")
idade = int(input("Digite sua idade: "))
print("Seu nome é %s e sua idade é %d" % (nome, idade))
```

Salve o programa e navegue até o diretório onde você salvou o programa. Clique com o botão direito em algum lugar dentro do diretório e escolha a opção *Abrir no terminal.* Agora execute o seu programa com o comando:

python3 nomeDoSeuPrograma.py

Identação e blocos de códigos

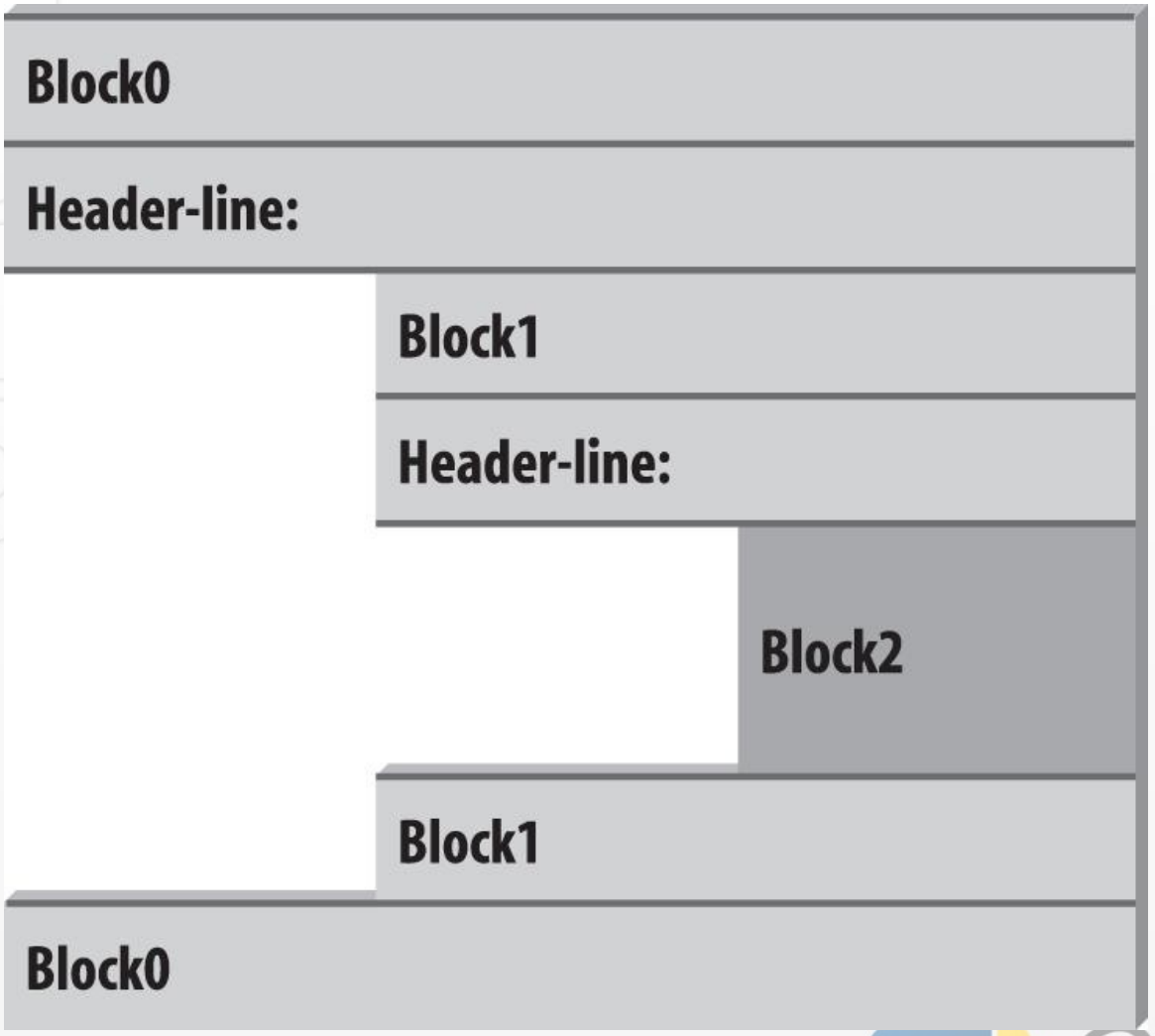
O Python executa instruções, uma após a outra, do topo até o final do arquivo, a menos que você diga o contrário.

Os limites dos bloco e instruções são detectados automagicamente.

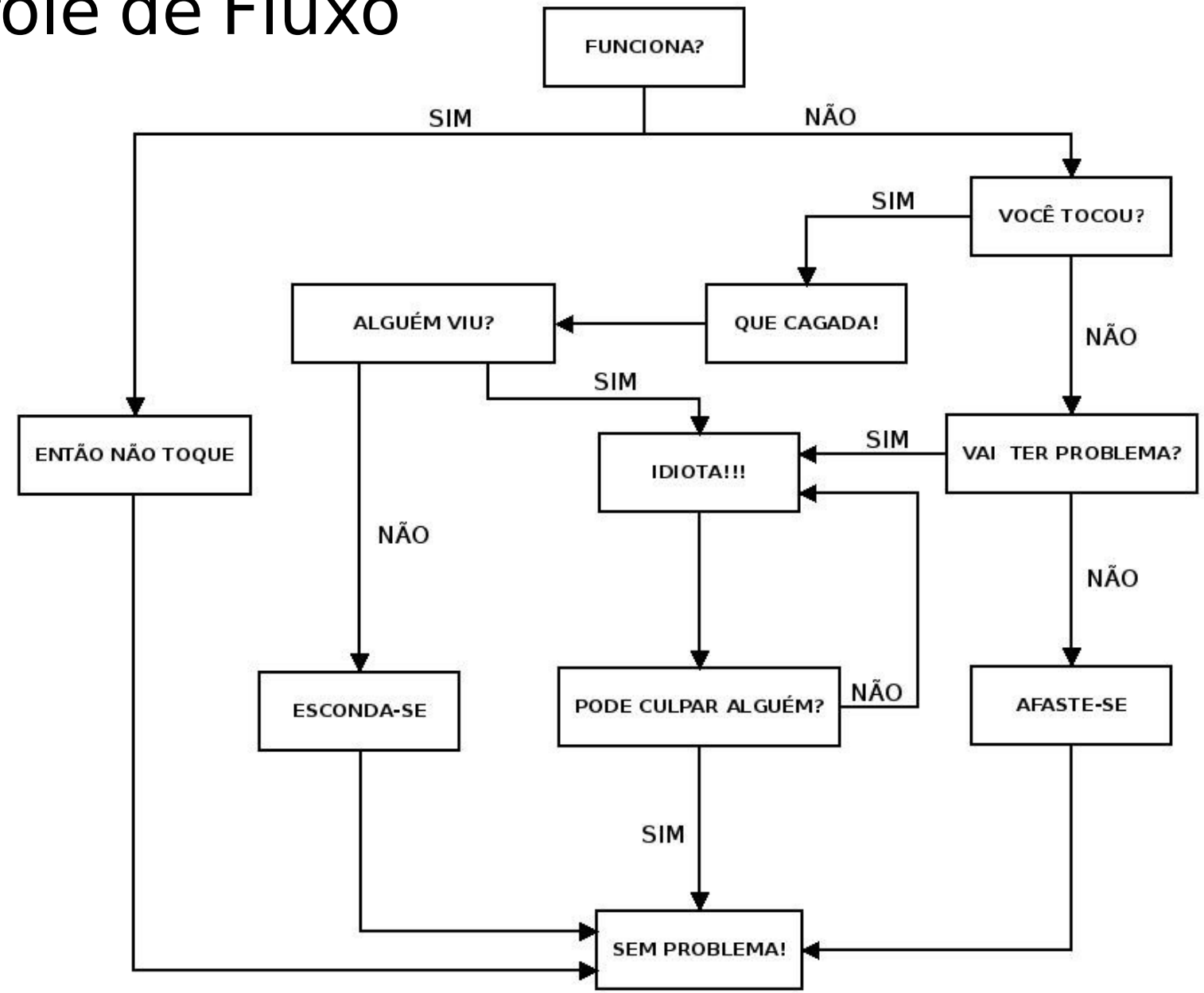
Bloco de instrução = cabeçalho + ” : ” + instruções identadas

Linhas vazias, espaços e comentários, geralmente são ignorados.

Identação e blocos de códigos



Controle de Fluxo



Controle de Fluxo

A instrução *if* permite desviar o fluxo do programa, checando um ou mais condições e alterando o comportamento do programa de acordo com elas. Vejamos um exemplo simples:

```
nome = input("Digite seu nome: ")
idade = int(input("Digite sua idade: "))

if idade >= 18:
    print("%s, você tem 18 anos ou mais." % nome)
```

Se idade for maior ou igual a 18, uma mensagem será exibida, do contrário, a execução pula a instrução *print()* e o programa é encerrado.

Controle de Fluxo

Junto com o *if,* podemos utilizar a instrução *else* para construir um lógica *se / senão*:

```
nome = input("Digite seu nome: ")
idade = int(input("Digite sua idade: "))

if idade >= 18:
    print("%s, você tem 18 anos ou mais." % nome)
else:
    print("%s, você ainda não tem 18 anos." % nome)
```

Nota: também é possível utilizar o *else* com outras instruções, com o for e try / except
http://psung.blogspot.com.br/2007/12/for-else-in-python.html
http://python-notes.curiousefficiency.org/en/latest/python_concepts/break_else.html

Controle de Fluxo

Existe também a instrução *elif*, ela somente faz a verificação e executa a instrução do seu bloco se a condição do *if* não for satisfeita:

```
var = 'py'

if var in 'python':
    print('passei pelo if')

elif var in 'numpy':
    print('passei pelo elif')

else:
    print('passei pelo else')
```

Controle de Fluxo

Com uma sequência de *if* teremos um resultado diferente:

```
var = 'py'

if var in 'python':
    print('passou pelo primeiro if')

if var in 'numpy':
    print('passou pelo segundo if')

else:
    print('passou pelo else')
```

Note que são verificados todos as instruções *if*, independente do resultado da comparação do if anterior.

Controle de Fluxo

Note também que se a condição do primeiro *if* não for satisfeita, a instrução *else* não é chamada, pois ela pertence ao bloco do segundo *if*.

```
var = 'py'

if var in 'linux':
    print('passou pelo primeiro if')

if var in 'numpy':
    print('passou pelo segundo if')

else:
    print('passou pelo else')
```

Estruturas de Repetição

A expressão *for* é projetada para percorrer os item de uma sequência ou por um objeto iterável e rodar um bloco de código para cada iteração.

```
>>> for item in sequência:
...         bloco de instrução que pode ou não
...         lidar com o item atual da iteração
```

É possível iterar sobre strings, listas, tuplas, dicionários* e outro objetos iteráveis que não teremos tempo de abordar.

* Em dicionários, o *for* percorre as chaves, para que no bloco de instrução elas estejam disponíveis para as chamadas dos valores na sintaxe dic[*chave*]

Estruturas de Repetição

```
>>> lista = ['pyhton', 'linux', 'flisol', 2016]
>>> for elem in lista:
...     print(elem)
...
pyhton
linux
flisol
2016

>>> dic = {'Oficina': 'Python', 'Linux': 'Software Livre', 'FLISoL': 2016}
>>> for key in dic:
...     print(dic[key])
...
2016
Python
Software Livre
```

python

Estruturas de Repetição

```
>>> for key in dic:
...     print(key)
...
FLISoL
Oficina
Linux

>>> for letra in 'Python':
...     print(letra)
...
P
y
t
h
o
n
```

Estruturas de Repetição

Para gerar uma sequência numérica podemos utilizar a função range() passando pelo menos um argumento para ela:

```
range( início, fim, passo )
```

Inclui o valor passado para o início, mas não o valor final.

```
>>> for num in range(2,8):
...     print(num, end=' ')
...
2 3 4 5 6 7

>>> for num in range(0, 20, 3):
...    print(num, end=' ')
...
0 3 6 9 12 15 18
```

Estruturas de Repetição

Python tem uma função bastante útil quando precisamos obter a posição de cada item de uma sequência junto com o item:

```
>>> lista = ['pyhton', 'linux', 'flisol', 2016]
>>> for pos, item in enumerate(lista):
...     print(pos, " ---> ", item)
...
0  --->  pyhton
1  --->  linux
2  --->  flisol
3  --->  2016
```

Estruturas de Repetição

Podemos também aninhar um bloco *for* dentro de outro bloco *for:*

```
>>> lista = [ ('Pyhton', 'Linux', 'Flisol', 2016),
              ('UNISAL', 'UNESP'),
              ('Oficina', 'Programação', 'Lógica')]

>>> for tupla in lista:
...         for item in tupla:
...             print(str(item) + '\t', end=' ')
...         print('\n')
...
Pyhton   Linux   Flisol   2016

UNISAL   UNESP

Oficina   Programação   Lógica
```

Estruturas de Repetição

A instrução *while* executa um bloco repetidamente, enquanto uma condição for verdadeira, mantendo o bloco num laço (ou loop):

```
>>> s = 'Python'
>>> while s:
...         print(s, end='   ')
...         s = s[1:]
...
Python   ython   thon   hon   on   n
```

Essa instrução *while* é equivalente a *while s != '',* ou seja, enquanto s for diferente de uma string vazia, execute o bloco.

Estruturas de Repetição

Outro modo de usar o *while,* é utilizar um contador que vai sendo incrementado ou decrementado a cada laço:

```
>>> a=0; b=10
>>> while a < b:
...         print(a, end=' ')
...         a += 1
...
0 1 2 3 4 5 6 7 8 9
```

Adivinhe o número

Vamos escrever um jogo que peça ao usuário chutar um número entre 1 e 50. O programa irá verificar se esse valor é menor, maior ou igual ao valor gerado aleatoriamente e retornará uma resposta adequada ao jogador. O jogador terá 10 chances para acertar o número.

Abra o arquivo *adivinhe-o-numero.py*

Funções

Funções são um forma de empacotar uma lógica pra ser usada em um programa em mais de um lugar e mais de uma vez.
A sintaxe para *definir* uma função é:

def nome_da_funcao **():**
    ...bloco de instrução

Para chamar a função dentro do seu código, basta declarar o nome da função seguido de ( )

```
>>> def imprime_nome():
...         nome = input("Digite seu nome: ")
...         print(nome)

>>>imprime_nome()
Digite seu nome:
```

Funções

Funções podem receber argumentos:

```
>>> def dobro(x):
...         print(x * 2)
...
>>> dobro(8)
16
```

Funções podem retornar valores:

```
>>> def dobro(x):
...         return x * 2
...
>>> num = dobro(8)
>>> num
16
```

Funções

Variáveis declaradas dentro do bloco de instruções de uma função, serão variáveis locais, do escopo da função.

```
>>>num = 42

>>>def mult(x,y):
...         num = x * y
...         print(num)
...
>>> mult(3, 2)
6
>>> num
42
```

Funções

Funções podem ser atribuídas a variáveis:

```
>>> def dobro(x):
...     return x * 2
...
>>> db = dobro
>>> db
<function dobro at 0xb736fa04>

>>> db(4)
8

>>> db = dobro()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: dobro() missing 1 required positional argument: 'x'
```

Cifra de César

Vamos escrever um programa para criptografar e decriptografar uma mensagem seguindo a ideia do Imperador César. O programa terá 3 funções e mais um função principal, que será no final do código. Ela utilizará as outras 3 funções para processar a mensagem do usuário. As outras 3 funções serão:

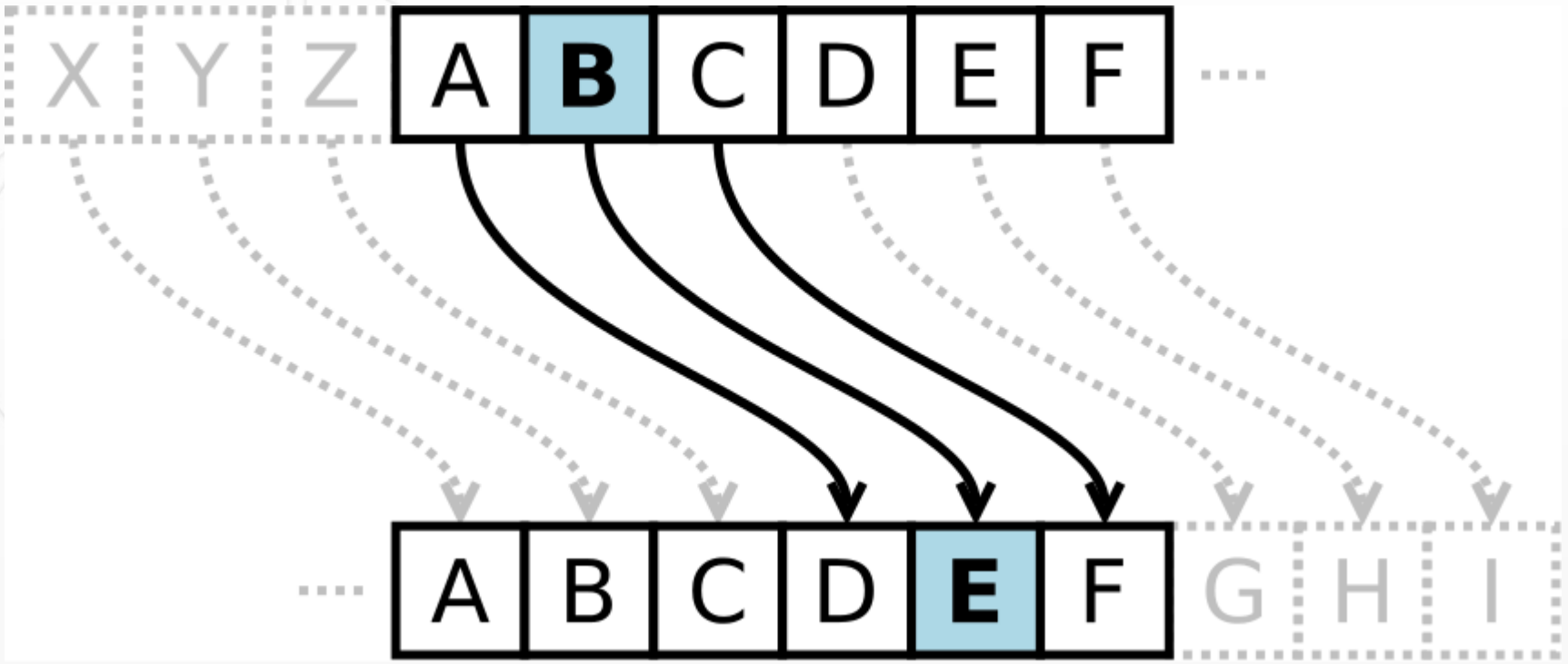
'recebeModo' - pergunta se o usuário quer criptografar ou decriptografar e garante que uma entrada válida foi recebida.

'recebeChave' - pede o valor da chave para o usuário e devolve esse valor caso ele seja adequado.

'geraMsgTraduzida' - traduz a mensagem do usuário trocando cada caractere por caractere correspondente à chave passada pelo usuário.

Abra o arquivo *cifra-de-cesar.py*

Cifra de César



python

tkinter

É uma biblioteca padrão do Python com ferramentas para a criação de uma interface gráfica (GUI - Graphic User Interface).

```
>>> from tkinter import Tk, Label
>>> window = Tk()
>>> widget = Label(window, text='Hello GUI world!')
>>> widget.pack()
```

Para rodar esse script a partir de um arquivo, é preciso chamar o método *.mainloop()* no final no arquivo.

```
from tkinter import Tk, Label
window = Tk()
widget = Label(window, text='Hello GUI world!')
widget.pack()
window.mainloop()
```

YouTube Downloader

Chegou a hora fazer um programa pra chamar de seu e com uma aplicação no mundo real. Vamos criar uma interface gráfica para um script que faz download de vídeos do YouTube.

import this

Uma breve introdução à linguagem python

python

tkinter

Abra o arquivo *youtube-downloader-gui.py*

Atribua a uma variável chamada *window* uma instância de *Tk.* Isso fará com que o Python crie uma janela principal. Agora vamos dar um título para o programa, ajustar o tamanho da janela e impedir que o usuário redimensione nossa janela.

Utilize o método *.title()* da instância *window*, passando uma *string* como argumento.

Sugestão: “Youtube Downloader”

Na próxima linha chame o método *.geometry()* com o argumento *'500x200'* e o método *.resizable()* com os argumentos *“0,0”*

zero vírgula zero

tkinter

Vamos exibir um título para o nosso programa. O widget *Label* implementa uma caixa de display onde você pode colocar textos ou imagens.

Atribua a uma variável *title* uma instância de *Label*, passando o seguinte argumento para *Label*

window, text='Youtube Downloader', font=('Arial', 25), fg='Blue'

Na próxima linha, chame o método .pack() da instância *title.*

Na linha seguinte, crie uma outra instância de *Label* chamada *msg*, com os argumentos

window, text='', font=('Arial', 15)

tkinter

Vamos criar uma entrada de texto para o usuário inserir o link do vídeo e um nome de saída do arquivo após o download. Usaremos o widget *Entry,* que aceita uma única linha de texto inserida pelo usuário.

Atribua uma instância de *Entry* a uma variável chamada *entry_url*, com os seguintes parâmetros:

window, width=60, justify='center'

Nas próximas três linhas vamos utilizar alguns métodos de *Entry*, o *.insert(), .focus_set()* e o *.pack().* Passe o seguinte argumento para o método *insert:*

0, 'Cole aqui a URL'

O valor zero e a string que irá ser exibida dentro da caixa de texto

tkinter

Precisamos receber o nome de saída do vídeo. Crie uma variável chamada *entry_name_video*, ela também será uma instância de *Entry* com os mesmos argumentos de *entry_url*. Nas próximas duas linhas, chame o método .*insert()* com o argumento

0, 'Digite um nome para o vídeo'

e o método pack().

tkinter

Vamos definir uma função que não receberá nenhum argumento, chamada *click_button.* Ela irá responder ao evento do clique do mouse.

Entry possui um método *.get()*, que captura a entrada de texto feito pelo usuário. Dentro de *click_button*, com o método *.get()* vamos colocar as entradas que *entry_url* e *entry_name_video* receberão dentro das variáveis *url* e *name_video*

tkinter

Agora vamos implementar um bloco *if / else*

```
if download_video(url, name_video):
```

Se a função retornar *True*, vamos alterar algumas propriedades de *msg* (que é uma instância do widget *Label*)

```
        msg['fg'] = 'Green'
        msg['text'] = 'Download feito com sucesso!'

    else:
        #Red
        #Ocorreu alguma falha
```

tkinter

Agora criaremos um botão, usaremos o widget *Button*, que permite exibir textos ou imagens nos botões. Podemos fazer com que o botões chamem funções ou métodos quando receberem um clique.

Atribua uma instância de *Button* a uma variável chamada *btn*, com os seguintes parâmetros:

window, text='Download now', width=20, command=click_button

Na próxima linha chame o método .pack() da instância *btn* e na linha seguinte chame o mesmo método para instância *msg*

Na última linha chame o método *.mainloop()* da instância *window*

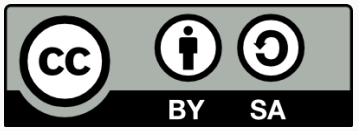
Use GNU/Linux, softwares livres, programe em Python e muito obrigado pela atenção!!

python

import this

Uma breve introdução à linguagem python

Você pode compartilhar, alterar e gerar novos materiais a partir desse, mas sempre citando a fonte e distribuindo sobre essa mesma licença.



https://creativecommons.org/licenses/by-sa/3.0/br/

python