



PARTE 1

C O
D E



REVISÃO

Entrada: Receber dados do teclado, de um arquivo, da rede ou de algum outro dispositivo;

Saída: Exibir dados na tela, salvá-los em um arquivo, enviá-los via rede, etc;

Matemática: Executar operações matemáticas básicas como adição e multiplicação;

Execução condicional: Verificar a existência de certas condições e executar o código adequado;

Repetição: Executar várias vezes alguma ação, normalmente com alguma variação.

Processamento: Resultado das Entradas de Dados.

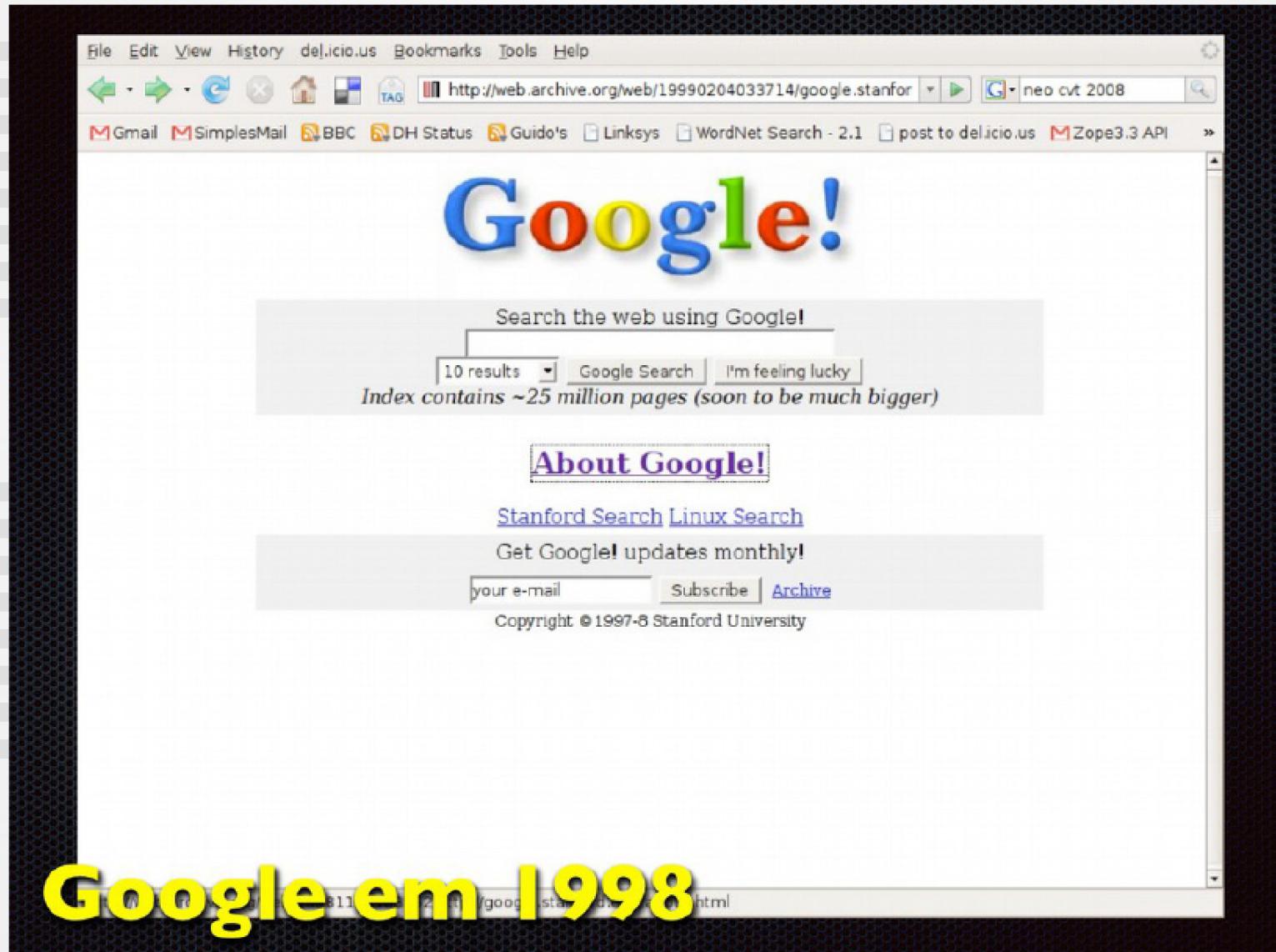
POR QUE O NOME PYTHON?

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](#)

POR QUE PYTHON?



Google em 1998

POR QUE PYTHON?

The screenshot shows a web browser window with a dark theme. The title bar reads "File Edit View History del.icio.us Bookmarks Tools Help" and the address bar shows "http://web.archive.org/web/19990204033714/google.stanford". The main content is the "About Google!" page. It features the iconic Google logo and sections for "Links" and "Credits". The "Credits" section lists various contributors and organizations, including a mention of the "Computer Science Department at Stanford University". A red arrow points to this specific link. At the bottom, it says "Last modified: Sun Sep 27 20:26:46 PDT 1998" and "http://web.archive.org/web/19990204033714/http://www.python.org/".

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](#)

Last modified: Sun Sep 27 20:26:46 PDT 1998

<http://web.archive.org/web/19990204033714/http://www.python.org/>

POR QUE PYTHON?

The image is a composite of several elements. On the left, there's a dark rectangular area containing the Industrial Light & Magic logo (a stylized lightbulb icon) and the text "INDUSTRIAL LIGHT & MAGIC". Below this are two menu items: "ABOUT ILM" and "OUR WORK". To the right of this dark area is a movie screenshot from Iron Man 2. It shows Tony Stark in his red Iron Man suit and Rhodey in his blue War Machine suit, both looking forward with serious expressions. The scene is set in a dark, industrial-looking environment. In the bottom right corner of the screenshot, the text "IRON MAN 2" is visible. The entire composite image is set against a dark background with a vertical decorative element on the far left consisting of several horizontal bars of varying lengths.

INDUSTRIAL
LIGHT & MAGIC

ABOUT ILM

OUR WORK

IRON MAN 2

INDUSTRIAL
LIGHT & MAGIC

LUCASARTS

LUCASFILM
ANIMATION

LUCASFILM
LTD.

LUCAS LICENSING

LUCAS ONLINE

SKYWALKER
SOUND

Por que Python?

Por que o Nubank sempre busca
cientistas de dados e paga até R\$ 25 mil

Encontre o ERRO

VENHA TRABALHAR COM A GENTE

Desenvolvedor Flutter Sr.

São Paulo - SP

Para atuação em desenvolvimento e manutenção de Apps existentes.

Atuação com:

- Desenvolvimento mobile em Flutter com Dart;
- PostgreSQL;
- Angular 6;

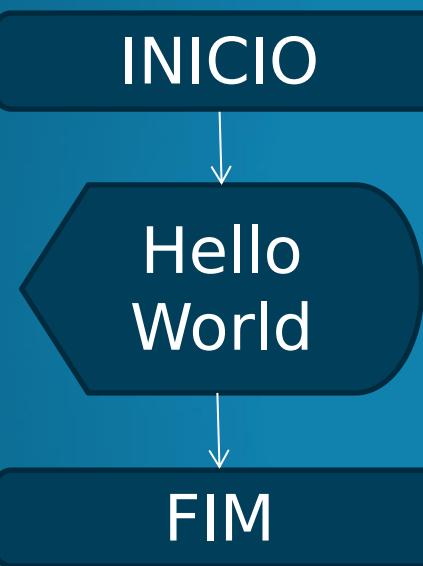
 Envie seu currículo
rh@fabricads.com.br

 **FábricaDS**
SOLUÇÕES EM TECNOLOGIA

Homologada em Dez/18

Por que Python?

FLUXOGRAMA



ALGORITMO

1. INICIO
2. EXIBIR “Hello World”
3. FIM

PYTHON

```
allan@h4ck3r-mrrobot:~$ python3.7
Python 3.7.3 (default, Mar 26 2019, 01:59:45)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

Simplicidade.
Sintaxe próxima ao
pseudo código.

Por que Python?

- Que permite focar nos algoritmos, que é o importante;
- APRENDIZADO PROGRESSIVO
- Variáveis > Operadores > Condicionais > funções > tipos > Laços
- É possível passar de forma quase atômica, sem ter que expor estudantes a conceitos avançados.
- Facilitando assim o aprendizado de outras linguagens
- Se já ajuda o profissional no dia-a-dia, pra quem está aprendendo faz toda diferença.
- Utilizar Python para aprender Algoritmos
- Aprender com Python.

Por que Python?

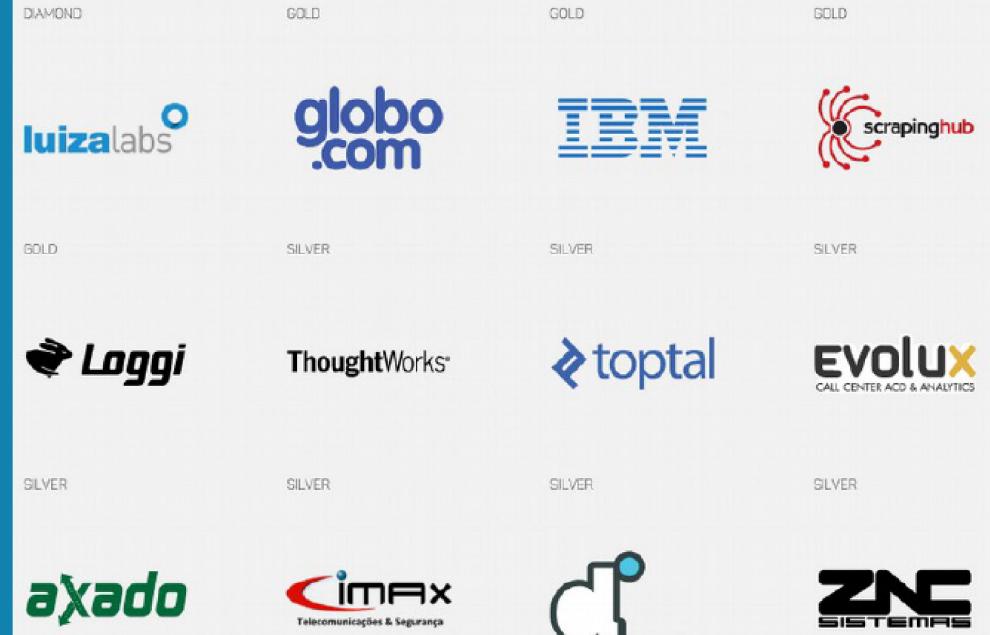
SPONSORS



PYTHONBRASIL 2015

SOBRE PALESTRANTES LOCAL PROGRAMAÇÃO PATROCINADORES

PATROCINADORES



POR QUE 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	try	

31 Palavras



Aprenda PYTHON



Universidade
PYTHON

COMUNIDADE

- **URL:** <http://python.org/>
- **Mail:** python-list@python.org, python-help@python.org
- **IRC:** irc.freenode.org, **canal #python**

- **URL:** <http://www.pythonbrasil.com.br/>
- **Mail:** python-brasil@yahoo(grupos).com.br
- **IRC:** irc.freenode.org, **canal #python-br**

INSTALL PYTHON



python
powered

```
print("Hello, world!")
```

TRADIÇÕES NUNCA DEVEM MORRER

```
allan@h4ck3r-mrrobot:~$ python3.7
Python 3.7.3 (default, Mar 26 2019, 01:59:45)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> print("Hello World")
Hello World
>>>
>>> █
```

import this



The Zen of Python, by Tim Peter

```
>>> 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, by Tim Peter

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.

Diante 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!

TIPO DE DADOS

```
>>> a=15  
>>> b=5  
>>> a=15.0  
>>> b=5.0
```

```
>>>a=15  
>>>type(a)  
>>>b=5  
>>>type(b)
```

TIPO DE DADOS

```
>>> a=True  
>>> type(a)  
>>> a='Turma'  
>>> b='Python'  
>>>a+b  
>>>print(a+b)
```

EXIBIR INFORMAÇÕES

```
>>> print('Eu sou F...')
```

ORGANIZAÇÃO

Em *Python* não existe um delimitador específico para blocos de código. A delimitação é feita pela identação:

```
print "O valor de a é "
if a == 0:
    print "zero"
    a = "zero"
else:
    print a

exit()
```

Isto garante que o código seja sempre legível.

TIPOS DE DADOS

Além dos tipos básicos (inteiro, ponto flutuante, ...), o *Python* possui ainda outros tipos de mais alto nível:

Listas []: como um vetor em outras linguagens, lista é um conjunto de valores acessados por um índice numérico, inteiro, iniciado por zero. Em *Python*, uma lista ainda podem armazenar todo tipo de valores.

Tuplas: *Tuplas* são sequências de elementos arbitrários como listas, com a exceção de que são imutáveis.

Strings: *string* em *Python* é uma sequência imutável, alocada dinamicamente e sem restrição de tamanho.

Dicionários: dicionários são sequências que podem utilizar índices (imutáveis) de tipos variados. conhecidos como *arras associativos*.

Arquivo: *Python* possui um tipo pré-definido para manipular arquivos. Este tipo permite que o arquivo seja facilmente lido, alterado e escrito.

Classes e Instâncias: classes são estruturas especiais que servem para apoiar programação orientada a objetos. Instâncias são expressões concretas destas classes.

ORIENTAÇÃO A OBJETO

Em *Python*, todos os dados podem ser considerados objetos.

COMENTÁRIOS

Comentários em *Python* seguem a mesma estrutura dos comentários em *bash script*:

```
>>> # isto é um comentário
```

HELLO.PY

```
print("Ola Mundo")
```

```
python3.7 hello.py
```

TIPOS DE DADOS

O Python possui alguns tipos numéricos pré-definidos: inteiros (*int*), ponto flutuante (*float*), booleanos (*bool*) e complexos (*complex*). Este tipos suportam as operações matemáticas básicas.

```
>>> a, b = 1, 2.5          #Atribui o valor 1 a “a” e o valor 2,5 para “b”
>>> c = True                #Booleano
>>> z = 3 + 4j              #Complexo
>>> a+b                    #soma, o resultado é em exibido em ponto flutuante
>>> int(a+b)                #Resultado Inteiro
>>> b * z                  #Resultado Complexo
>>> type(z)                 #Tipo de Dado
```

TIPOS DE DADOS

Python também trabalha com base octal, binário e hexadecimal.

```
>>> a = 0o11          #OCTAL  
>>> b = 0b1001       #Binário  
>>> c = 0x001        #Complexo
```

OPERAÇÕES MATEMÁTICAS

```
>>> 5+2          #adição  
>>> 5-2          #subtração  
>>> 5*2          #multiplicação  
>>> 5/2          #divisão  
>>> 5//2         #divisão inteira  
>>> 5**2         #potencia  
>>> 5%2          #resto da divisão
```

LISTAS []

Lista é uma sequência de valores indexadas por um inteiro.
Uma lista pode conter qualquer tipo de valor, incluindo
valores de tipos mistos:

```
>>> numeros = [1, 2, 3]
>>> nomes = [ 'alberto', 'carlos', 'simone' ]
>>> misto = [ 1,2,3.0,'alberto','carlos',3.5,'simone' ]
>>> listas = [ numeros, nomes, misto ]
>>> print(listas)
[[1, 2, 3], ['alberto', 'carlos', 'simone'], [1, 2, 3.0,
'alberto', 'carlos', 3.5, 'simone']]
```

LISTAS []

Os elementos da lista podem ser acessados por meio de índices que vão de 0 até o comprimento da lista -1:

```
>>>len(numeros)
>>>len(numeros)-1
>>>numeros[0]
>>>numeros[2]
>>>listas[0]
>>>listas[1]
>>>listas[1][1]
```

LISTAS [] - SELEÇÕES

Python também permite acessar uma lista de trás para frente, identificando o índice por um sinal de menos:

```
>>>numeros[-1]      #último valor  
>>>numeros[-2]      #penúltimo valor
```

Fatias, ou *slices*, de uma lista podem ser geradas facilmente com o “:”

```
>>>n = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 ]  
>>>n[2:4]  
>>>n[6:]  
>>>n[5:-1]
```

LISTAS [] - MÉTODOS

Como foi dito anteriormente, em *Python* tudo é um objeto.
Vamos ver alguns métodos de uma lista:

```
>>>numeros.append(0)
>>>numeros
>>>numeros.sort()
>>>numeros
>>>numeros.reverse()
>>>numeros
>>>numeros.pop()
>>>numeros
```

LISTAS [] - MÉTODOS

Outros métodos de listas:

```
>>>n = [1,2,3]
>>>m = [4,5,6]
>>>n
>>>m
>>>n;m
>>>n.extend(m)
>>>n.insert(2,'galo'); n
>>>n.remove(4);n
>>>n.append('barco');n
>>>n.pop()
>>>n.pop(2)
```

LISTAS [] - MÉTODOS

Outros métodos de listas:

```
>>>n.index(5)
>>>n.append(5)
>>>n.extend(m)
>>>n
>>>n.count(5)
>>>n.sort() ; n
>>>n.reverse(); n
>>>n.remove(4);n
```

TUPLAS ()

Tuplas são como listas, um conjunto de elementos acessíveis por um índice inteiro.

LISTAS

```
>>>l = [1,2,3]  
>>>l  
>>>l[0]='laranja';n
```

TUPLAS

```
>>>t = (1,2,3)  
>>>t  
>>>t[0]  
>>>t[0] = 'laranja'
```

STRINGS

String é uma sequência imutável com o propósito especial de armazenar cadeias de caracteres.

```
>>>a = 'abcdefg'  
>>>a[0]  
>>>a[-1]  
>>>a[6:]
```

```
>>>len(a)  
>>>a.__len__()  
>>>a.upper()
```

Você não consegue alterar um elemento da *string*

```
>>>a[1]='B'
```

STRING - ALGUMAS OPERAÇÕES

Colar strings (+)

```
>>>b = 'klmnopqrst'  
>>>a+b  
>>>print(a+b)  
>>>c = (a+b).upper()  
>>>print(c)
```

DICIONÁRIOS

Um dicionário é uma coleção de elementos, onde é possível utilizar um índice de qualquer tipo imutável.

```
>>>telefone = { "pedro":33231212,  
                 "patricia":42433545,  
                 "fernanda":88220091}  
>>>print(telefones["fernanda"])
```

DICIONÁRIOS - ALGUNS MÉTODOS

```
>>>telefone.keys()  
>>>telefone.values()  
>>>telefone.items()  
>>>telefone.__contains__('pedro')
```

OPERAÇÕES

Strings

```
>>>a='bits'  
>>>a*2  
>>>print('64'+a)
```

Listas

```
>>>a=[5,6,7,8]  
>>>b=[9,10]  
>>>print(b*2)  
>>>print(a+b)
```

Tuplas

```
>>>a=(2,3,4)  
>>>print(a+(5,6))
```

ATRIBUIÇÃO E CONDICIONAIS

```
>>>a=1  
>>>a+=1  
>>>print(a)  
>>>a*=10  
>>>print(a)  
>>>a/=2  
>>>print(a)  
>>>a,b = 3,5  
>>>a,b = b,a+b  
>>>print(a,b)
```

```
>>>a,b = 5,3  
>>>c = a if a > b else b  
>>>print(c)
```

```
>>>2 ==4  
>>>2 != 4  
>>>2 > 4  
>>>2 < 4  
>>>3 <= a
```

OPERADORES LÓGICOS

and, or, not

AND

True and True
True and False
False and True
False and False

OR

True or True
True or False
False or True
False or False

NOT

not True
not False

OPERADORES RELACIONAIS

Igual (==)
Diferente (!=)
Maior (>)
Maior igual (>=)
Menor (<)
Menor igual (<=)

COMBINANDO OPERADORES

```
>>>a, b = 5, 3  
>>>0 < a < b      #avaliação é feita da esquerda  
>>>0 < a > b      #para a direita  
  
>>>nome='pedro'  
>>>idade=25  
>>>nome=='pedro' and idade==25  
>>>len(nome) < 10 and idade > 30  
>>>len(nome) < 10 or idade > 30
```

ESTRUTURA DE CONTROLE IF

```
if condição:  
    #comandos  
    ...  
elif condição:  
    #comandos  
    ...  
else  
    #comandos  
    ...
```

```
a = 5; b = 8  
if a > b:  
    print("a é maior que b")  
    c = "maior"  
elif a == b:  
    print("a é igual a b")  
    c = "igual"  
else:  
    print("a é menor que b")  
    c = "menor"  
  
print(a,c,b)
```

ESTRUTURA DE CONTROLE IF

```
numero=5  
if numero % 2 == 0:  
    print('Par')  
else:  
    print('Impar')
```

```
numero=0  
if numero == 0:  
    print("Neutro")  
elif numero % 2 == 0:  
    print('Par')  
else:  
    print('Impar')
```

ESTRUTURA DE CONTROLE FOR

O laço for do *Python* é semelhante ao for do *bash*. Ele percorre uma sequência de elementos:

```
for variável in sequencia:  
    #comandos  
    ...
```

```
for numeros in range(1,10,2):  
    if numeros % 2 !=0:  
        print(numeros)
```

```
palavra='cachorro'  
for aux in palavra:  
    print(aux)
```

```
for numeros in range(10):  
    if numeros % 2 ==0:  
        print(numeros)
```

Como um programador joga baralho

O laço for do *Python* é semelhante ao for do *bash*. Ele percorre uma sequência de elementos:

```
naipes = 'copas ourtos espadas paus'.split()  
cartas = 'A 2 3 4 5 6 7 8 9 10 J Q K'.split()  
baralho = [(c,n) for n in naipes for c in cartas]  
print(baralho)  
print(len(baralho))
```

ESTRUTURA DE CONTROLE WHILE

O laço *while* é útil quando se é necessário fazer um teste a cada interação do laço. Assim como o *for*, aceita as instruções *continue* e *break*. Sua sintaxe completa tem a forma:

```
while condição:  
    #comandos  
    ...
```

```
numero=0  
while numero <= 20:  
    print(numero)  
    numero+=1
```

```
while 1:  
    nome = input("Digite um nome: ")  
    if nome == 'sair':  
        break  
    print("Obrigado")
```

EXCESSÕES

Com os laços *for* e *while*, e a condicionais *ifs*, todas as necessidades de controle em um programa podem ser implementadas. Mas quando algo inesperado ocorre, *Python* oferece uma forma adicional de controlar o fluxo de execução: a **exceção**

```
>>> a = [1, 2, 3]
>>> print(a[5])
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(a[5])
IndexError: list index out of range
```

A primeira linha anuncia que ocorreu um **traceback**. A segunda linha indica a linha de código e o arquivo onde o erro ocorreu (*stdin* – entrada padrão, modo interativo). Na terceira linha indica o tipo de exceção levantada - **IndexError**.

EXCESSÕES: Tratando exceções

```
>>> a = [1, 2, 3]
>>> try:
...     print(a[5])
... except IndexError:
...     print("Tentativa de acessar um índice inválido")
```

FUNÇÕES PRÉ-DEFINIDAS

Python possui várias funções pré-definidas, que não necessitam de importações externas. Vou passar rapidamente algumas destas funções:

range(a[, b[, c]]): retorna uma lista de inteiros de 0 a a-1, caso somente a seja passado como argumento; de a até b, caso a e b sejam passados como argumentos; a até b com o incremento c, caso a, b e c sejam passados como argumentos.

```
>>> range(10)      #gera uma lista com elementos de 0 a 9  
>>> range(3,10)    #gera uma lista com elementos de 3 a 9  
>>> range(30,3,-3) #gera uma lista com elementos de 30 a 4  
                      subtraindo de 3 em 3;
```

FUNÇÕES PRÉ-DEFINIDAS

```
>>> a = 'Alberto Santos do Dummont'
```

```
>>> len(a)
```

```
>>> a=[1,2,3]
```

```
>>> pow(3,2)
```

```
>>> pow(2,2)
```

```
>>> pow(3,2)
```

```
>>> round(5.48)
```

```
>>> round(5.548)
```

```
>>> round(5.548,1)
```

```
>>> for i in range(10): print(65+i)
```

FUNÇÕES PRÉ-DEFINIDAS

```
>>> min(1,6)
>>> min('a',2)
>>> min('abacate','flores')

>>> max(1,6)
>>> max('a',2)
>>> max('abacate','flores')

>>> abs(-3)
>>> abs(-3.0)

>>> hex(22)
>>> oct(22)
>>> hex(022)

>>> list('abacate')
>>> tuple('abacate')
```

FUNÇÕES PRÉ-DEFINIDAS

input([prompt]): lê qualquer coisa do teclado. Strings devem ser protegidas com aspas simples ou duplas.

```
>>> a = input("Entre com algo:")
```

FUNÇÕES

Sintaxe geral de uma função:

```
def nome (arg, arg, ... arg):  
    comando  
    ...  
    comando
```

```
>>> def f(x):  
...     return x*x  
>>> print f(10)
```

Listas, Vetores e Matrizes

Sintaxe geral de uma função:

```
>>>lista1 = [1, 2, 3, 4, 5]
>>>Vetor = [0,1,2,3]
>>>matriz2x2 = [[0, 1],[2, 3]]
>>>matriz3x2 = [[0, 1], [2, 3], [4, 5]]
>>>matriz2x3 = [[0, 1, 2], [3, 4, 5]]
>>>matriz=[[1,2],[0,-3]]
>>> print(matriz[0])
>>> print(matriz[0][0])
>>> print(matriz[0][1])
>>> print(matriz[1][0])
>>> print(matriz[1][1])
```

webs

