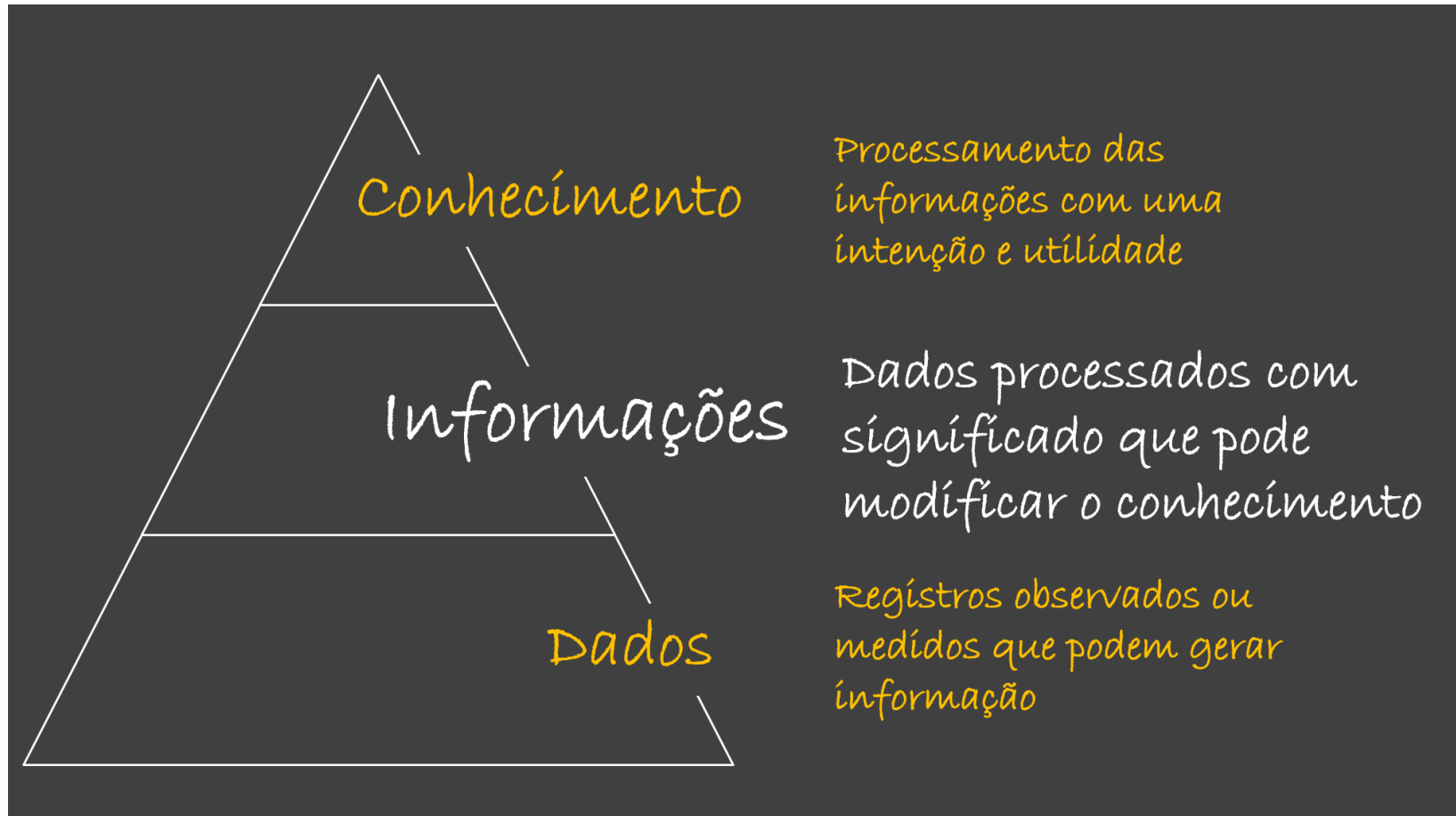
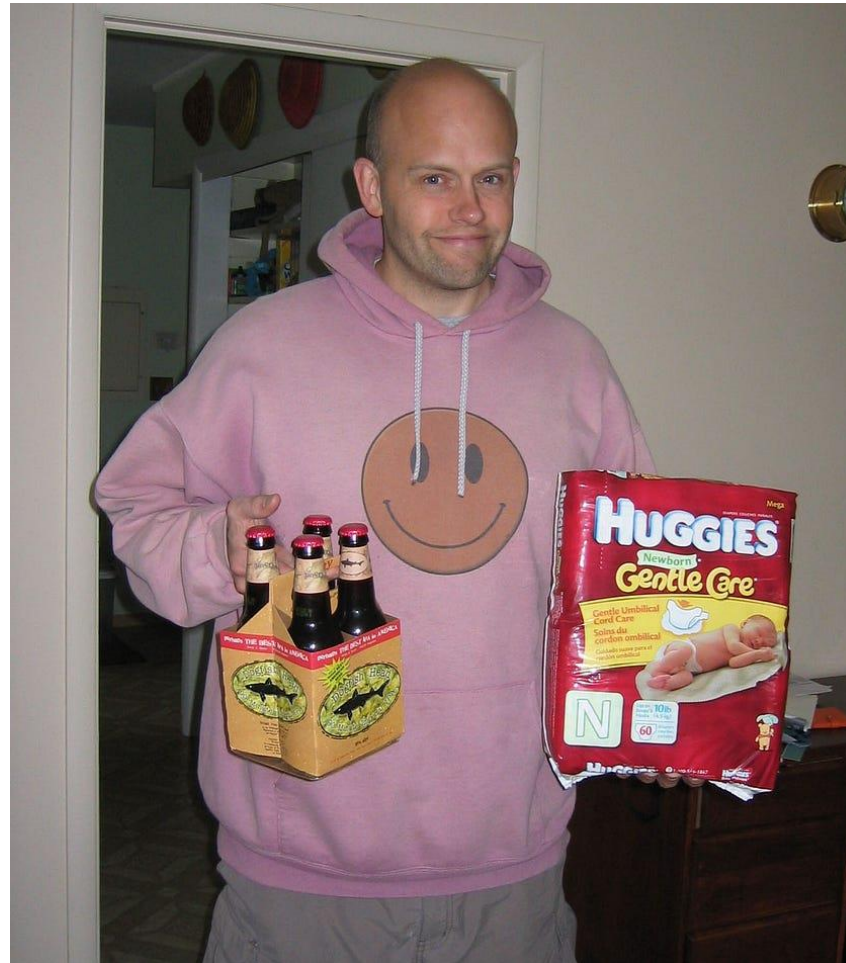


Aula 03







Market Basket Analysis



Customers Who Bought Echo (2nd Generation) - Smart speaker with Alex... Also Bought

Page 1 of 5



Echo Connect – requires...

★★★★☆ 1,442

\$34.99 ✓prime

Add to Cart



Kasa Smart Wi-Fi Plug...

★★★★☆ 12,775

\$19.99 ✓prime

Add to Cart



Etekcity 4 Pack Voltson...

★★★★☆ 1,359

\$39.99 ✓prime

Add to Cart



Portable Battery Base...

★★★★☆ 31

\$47.99 ✓prime

Add to Cart



$$\begin{aligned}
 \text{Rule: } X \Rightarrow Y & \begin{cases} \text{Support} = \frac{\text{freq}(X, Y)}{N} \\ \text{Confidence} = \frac{\text{freq}(X, Y)}{\text{freq}(X)} \\ \text{Lift} = \frac{\text{Support}}{\text{Supp}(X) \times \text{Supp}(Y)} \end{cases}
 \end{aligned}$$

Example:



Rule	Support	Confidence	Lift
$A \Rightarrow D$	2/5	2/3	10/9
$C \Rightarrow A$	2/5	2/4	5/6
$A \Rightarrow C$	2/5	2/3	5/6
$B \& C \Rightarrow D$	1/5	1/3	5/9

Rule: representa uma relação maior entre os dois conjuntos de itens

Suporte: A frequência relativa com que essa combinação ocorre no total das transações

Confidence: Mede a probabilidade de um item ser comprador, dado que outro já foi comprador,

<https://colab.research.google.com/drive/192ck68cnKvK4fvziPi9dWwhqNyxwUKuj?usp=sharing>



Data Mining Cases

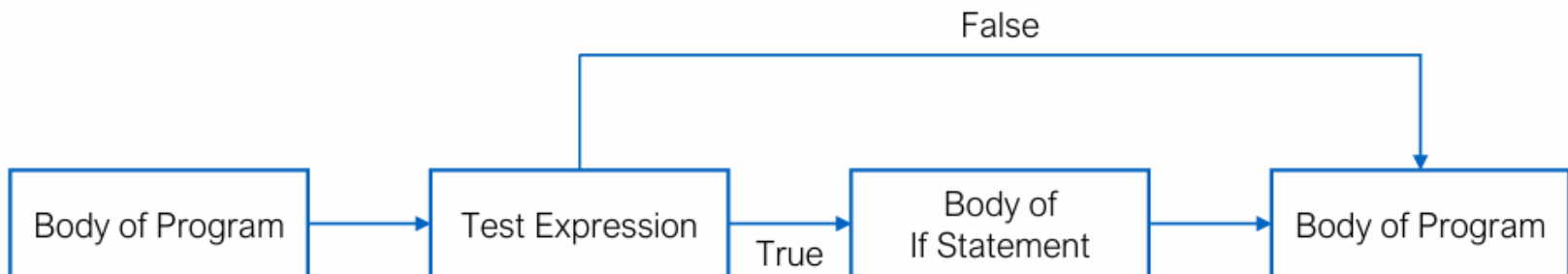


Voltando para Python...

Flow Control

If statement

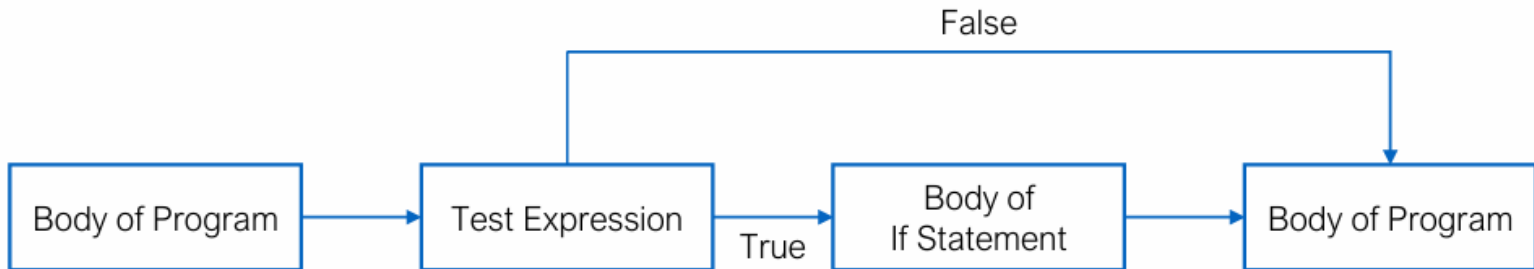
Um **If statement** permite efetuar flow control, criando uma condição na qual uma tarefa específica é executada se uma condição (test expression) for verdadeira



```
if x > y:  
    #body of if statement  
    print("x is greater than y")
```

If statement

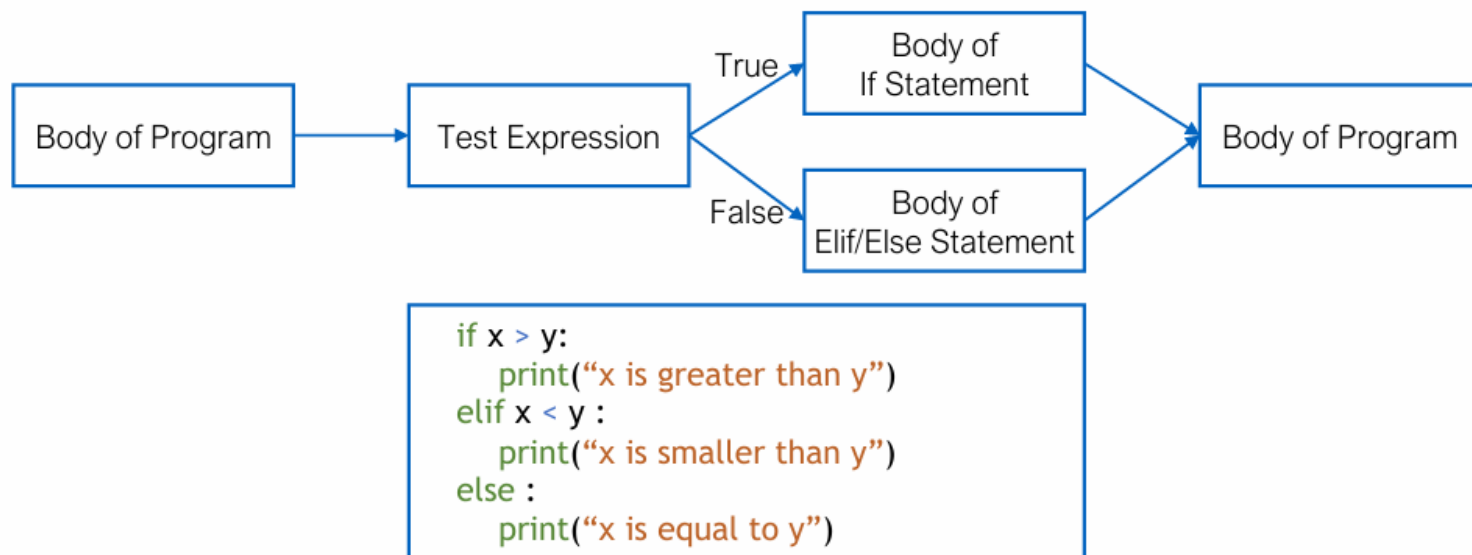
Um **If statement** permite efetuar flow control, criando uma condição na qual uma tarefa específica é executada se uma condição (test expression) for verdadeira.



Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

If statement

Um **If statement** permite efetuar flow control, criando uma condição na qual uma tarefa específica é executada se uma condição (test expression) for verdadeira. O **Else** e **Elif** permitem efetuar uma tarefa alteranativa.



If statement

Se só existir um comando para executar é possível poupar espaço escrevendo as instruções **if** e **else** numa só linha.

Com uma condição

```
if x > y: print("x is greater than y")
```

```
print("x is greater than y") if x > y
```

Com duas condições

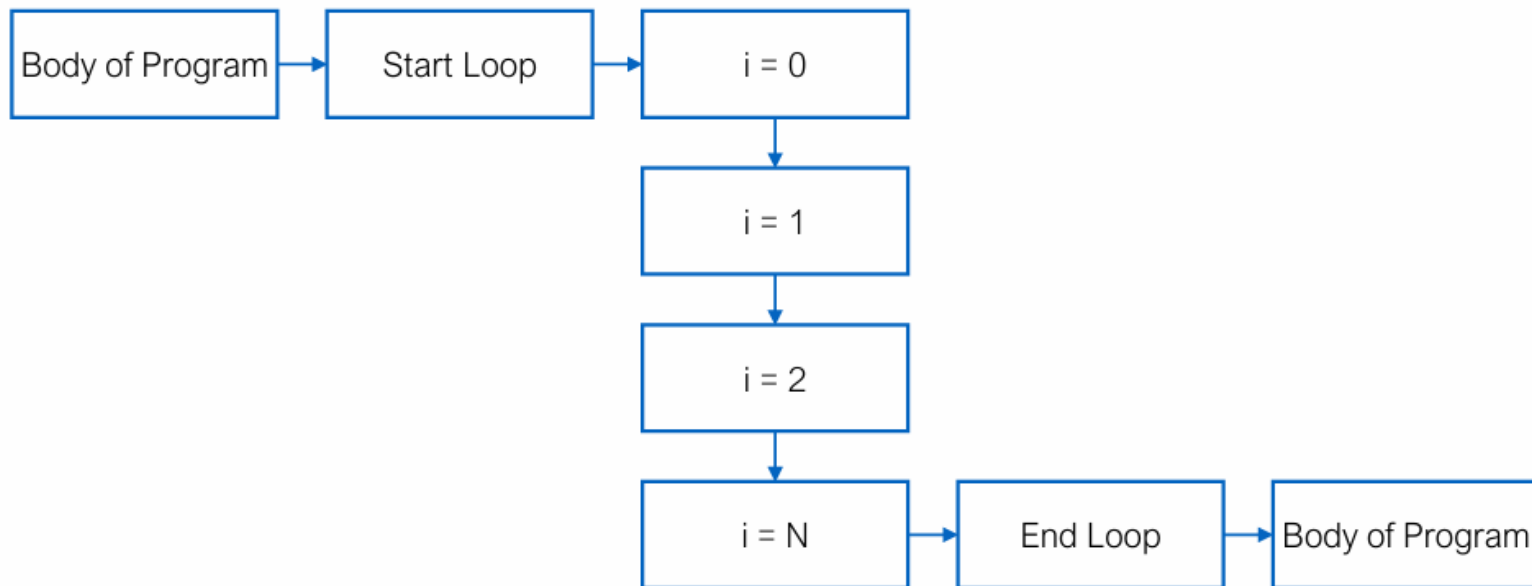
```
print("G") if x > y else print("L")
```

Com três condições

```
print("G") if x > y else print("E") if x == y else print("L")
```


Loops

O loop permite-nos iterar sobre uma sequência de elementos ou executar uma sequência de ações.



1. Controle do Loop

for:

- É usado quando **sabemos previamente** o número de iterações ou estamos iterando sobre uma sequência (como uma lista, tupla, string, ou range).
- O loop automaticamente gerencia o progresso e a iteração.

Exemplo:

```
python
for i in range(5):
    print(i)
```

Neste caso, sabemos que o loop será executado exatamente 5 vezes, e o valor de `i` será controlado automaticamente.

while:

- É usado quando **não sabemos o número exato de iterações** e dependemos de uma condição que pode mudar dinamicamente para terminar o loop.
- O programador precisa gerenciar a condição e garantir que ela eventualmente se torne `False` para evitar loops infinitos.

Exemplo:

```
python
x = 0
while x < 5:
    print(x)
    x += 1
```

Neste caso, o loop continua enquanto `x < 5`, e é nossa responsabilidade atualizar `x` dentro do loop.

2. Foco do Uso

Situação	Use <code>for</code>	Use <code>while</code>
Iterar sobre uma sequência	Sim (exemplo: listas, strings, range)	Não recomendado, mas possível
Condição baseada em lógica	Não recomendado	Sim
Número de iterações conhecido	Sim	Pode ser usado, mas é menos eficiente
Número de iterações desconhecido	Não recomendado	Sim

3. Exemplo Comparativo

Se quisermos iterar de 0 a 4:

Com **for**:

```
python
for i in range(5):
    print(i)
```

- Simple e direto.
- A contagem é gerenciada automaticamente.

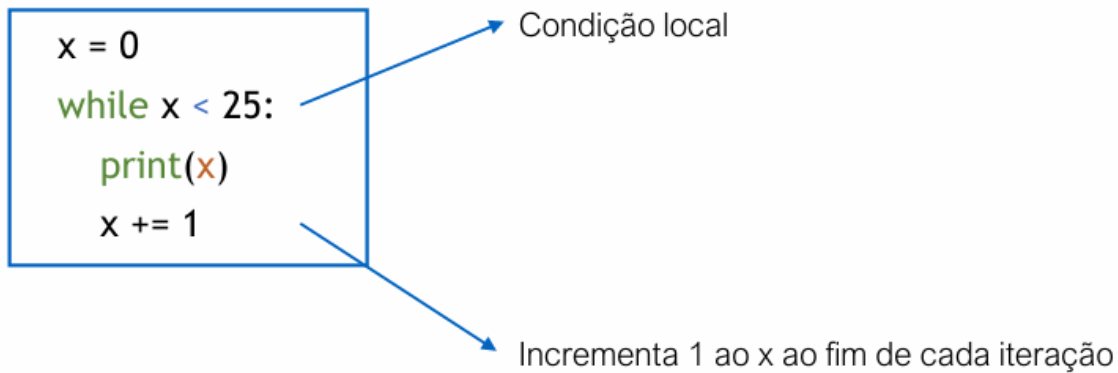
Com **while**:

```
python
x = 0
while x < 5:
    print(x)
    x += 1
```

- Mais detalhado: precisamos inicializar `x` e incrementá-lo manualmente.
- Se esquecermos o `x += 1`, o loop será infinito.

While Loop

O While Loop executa repetidamente uma ação enquanto uma condição se mantiver verdadeira.



While Loop

O While Loop executa repetidamente uma ação enquanto uma condição se mantiver verdadeira.

```
x = 0
while x < 25:
    print(x)
    x += 1
```

```
x = 0
while x < 25:
    print(x)
    if x == 3:
        break
    x += 1
```

O **break** para o while loop quando o x é igual a 3

While Loop

O While Loop executa repetidamente uma ação enquanto uma condição se mantiver verdadeira.

```
x = 0
while x < 25:
    print(x)
    x += 1
```

```
x = 0
while x < 25:
    print(x)
    if x == 3:
        break
    x += 1
```

```
x = 0
while x < 25:
    if x == 3:
        x += 1
        continue
    print(x)
    x += 1
```


Com o **continue** é possível
fazer skip de iterações

For Loop

O For Loop itera sobre os elementos de uma sequência (lista, tuples, dict, etc.)

```
words = ['Apple', 'Hollywood', 'Michael']
```

```
for word in words:  
    print(word)
```



A lista words neste contexto é um iterador

For Loop

O For Loop itera sobre os elementos de uma sequência (lista, tuples, dict, etc.)

```
words = ['Apple', 'Hollywood', 'Michael']
```

```
for word in words:  
    print(word)
```

```
for word in words:  
    if word == "Apple":  
        print(word)
```

```
for word in words:  
    if word == "Apple":  
        break  
    print(word)
```

```
for word in words:  
    if word == "Apple":  
        continue  
    print(word)
```


For Loop

O For Loop itera sobre os elementos de uma sequência (lista, tuples, dict, etc.)

```
for x in range(1,5):  
    print(x)
```

```
words = ['Banana', 'Hollywood', 'Michael']  
for i in range(len(words)):  
    print(words[i])
```

→ Length da lista **words**.

→ Devolve uma sequência de números que começa em 0 e termina no valor especificado.

Loops

O que esperas que aconteça?

```
while 1:  
    print("Where are we going?")
```

Loops

O que esperas que aconteça?

```
while 1:  
    print("Where are we going?")
```



Loops

O que é que este código produz?

```
>>> a = 5
>>> b = 1
>>> while a > 0:
    b *= a
    a -= 1
>>> print(b)
120
```

Loops

O que é que este código produz?

```
>>> a = 5
>>> b = 1
>>> while a > 0:
>>>     b *= a
>>>     a -= 1
>>> print(b)
120
```

Calcula o factorial de 5:

$$5! = 5 * 4 * 3 * 2 * 1$$

Nota: Indentação é importante!

Fazer loop dos números inteiros entre 0 e 99, imprimindo “par” se a divisão desse número por 2 tiver resto 0, ou “ímpar” se não tiver resto 0.

Opção A

```
for i in range(100):  
    if i % 2 == 0:  
        print("par")  
    else:  
        print("ímpar")
```

Opção B

```
for i in range(100):  
    if i % 2 == 0:  
        print("par")  
    else:  
        print("ímpar")
```

Opção C

```
for i in range(100):  
    if i % 2 == 0:  
        print("par")  
    else:  
        print("ímpar")
```

Comprehensions

Comprehensions

Como é que criamos uma lista de números?

```
lista = [0, 1, 2, 3, 4, ]  
print(lista)
```

Opção mais direta: definir um conjunto de elementos entre parênteses retos []

```
[0, 1, 2, 3, 4]
```

```
lista = []  
lista.append(0)  
lista.append(1)  
lista.append(2)  
lista.append(3)  
lista.append(4)  
print(lista)
```

Outra opção: definir uma lista vazia, e utilizar o método `append` para adicionar elementos à lista

```
[0, 1, 2, 3, 4]
```

Comprehensions

Como é que criamos uma lista de números?

```
lista = []  
for i in range(1,10):  
    lista.append(i)  
print(lista)
```

[1, 2, 3, 4, 5, 6, 7, 8, 9]

```
lista = []  
i = 1  
while i < 10:  
    lista.append(i)  
    i += 1  
print(lista)
```

[1, 2, 3, 4, 5, 6, 7, 8, 9]

Os loops oferecem uma flexibilidade extra que o método append não permite

Comprehensions

Como é que criamos uma lista de números?

```
lista = []  
for i in range(1,10):  
    lista.append(i**2)  
print(lista)
```

Lista com o quadrado dos números de 1 a 9

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

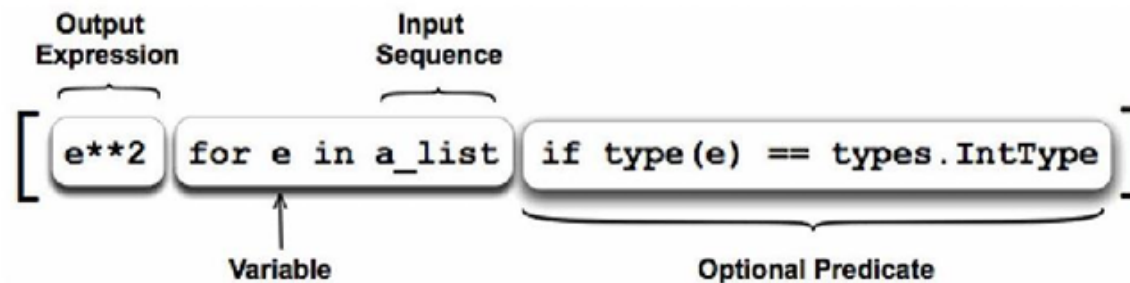
```
lista = [0,1]  
i = 2  
while i < 10:  
    i += 1  
    lista.append(lista[i-2] + lista[i-3])  
print(lista)
```

Sequência Fibonacci

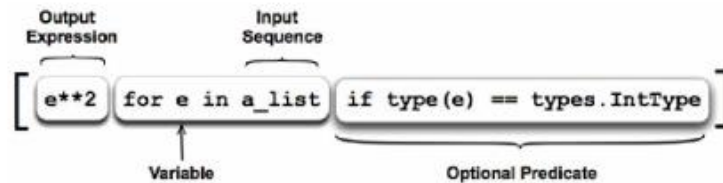
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Comprehensions

Podemos poupar linhas de código?



Comprehensions



```
lista = [i for i in range(1,10)]
print(lista)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
lista = [i**2 for i in range(1,10)]
print(lista)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
lista = [i for i in range(1,10) if i%2 == 0]
print(lista)
```

```
[2, 4, 6, 8]
```

Slices

Slices

Suponham que pretendem extrair uma sequência de números de uma lista.

Digamos, do 2º ao 5º elemento de uma lista, como é que se pode fazer isso?

Slices

Suponham que pretendem extrair uma sequência de números de uma lista.

Digamos, do 2º ao 5º elemento de uma lista, como é que se pode fazer isso?

```
a = [1,2,3,4,5,6,7,8,9,10]
b = []
for i in range(1,6):
    b.append(a[i])
print(a)
print(b)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[2, 3, 4, 5, 6]
```

Faz o trabalho, mas não é conveniente...

Slices

Suponham que pretendem extrair uma sequência de números de uma lista.

Digamos, do 2º ao 5º elemento de uma lista, como é que se pode fazer isso?

```
a = [1,2,3,4,5,6,7,8,9,10]
b = []
for i in range(1,6):
    b.append(a[i])
print(a)
print(b)
```

Faz o trabalho, mas não é conveniente...

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[2, 3, 4, 5, 6]
```

Existe uma forma mais direta?

Slices

Suponham que pretendem extrair uma sequência de números de uma lista.

Digamos, do 2º ao 5º elemento de uma lista, como é que se pode fazer isso?



Slices

Suponham que pretendem extrair uma sequência de números de uma lista.

Digamos, do 2º ao 5º elemento de uma lista, como é que se pode fazer isso?

```
a = [1,2,3,4,5,6,7,8,9,10]
```

```
a[1:6]
```

```
[2, 3, 4, 5, 6]
```

```
a[1:6:2]
```

```
[2, 4, 6]
```

Slices

i e j negativos são interpretados como $n + i$ e $n + j$, em que n é o número de elementos na dimensão correspondente. Um k negativo faz com que o passo se dirija para índices mais pequenos.

`list[i:j:k]`

```
a = [1,2,3,4,5,6,7,8,9,10]
```

```
a[-4:-2]
```

```
[7, 8]
```

Vamos praticar?

[Aula 03](#)

Próximas Aulas

1. Data Science Fundamentals

- › Data Science Definition /Applications
- › Data Mining main branches/techniques
- › Python Fundamentals – Key concepts
- › Data Understanding
- › Data loading and cleansing, Plotting, Feature extraction

Aula 1: Iniciando Big Data

- Introdução
- Computação distribuída vs paralela
- Apache Hadoop
- MapReduce e HFS
- Big Data cases

Aula 2: Spark e Ecosystema Big Data

- Ecosystema
- API: como usar e para que servem
- Pyspark, RDD, Dataframe API
- Databricks
- Projeto (Databricks para análise de dados)
- Hadoop vs Spark