

FIAP

NBA



Introdução ao Python

Dheny R. Fernandes

1. Introdução

1. Popularidade
2. Características
3. Ferramentas

2. Ambiente de Trabalho

1. Anaconda:

1. Jupyter Notebook

2. Google Colab

3. Sintaxe

1. Objetos
2. Controle de Fluxo
3. Laço de Repetição
4. Match

4. Tipos de Dados

1. Número
2. Texto
3. Lista
4. Tupla
5. Dicionário

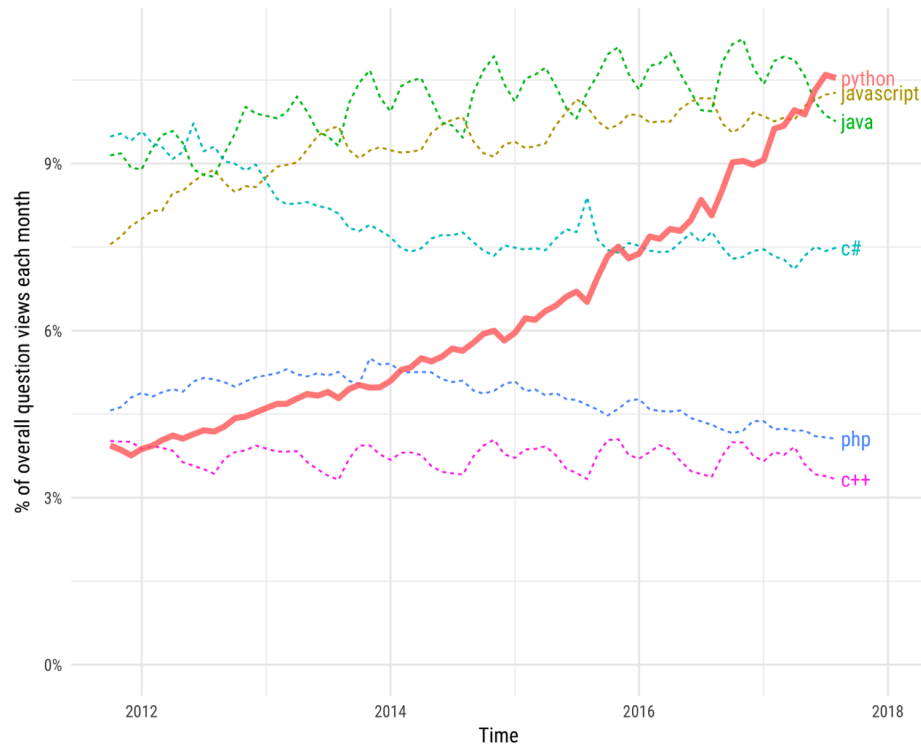
Introdução



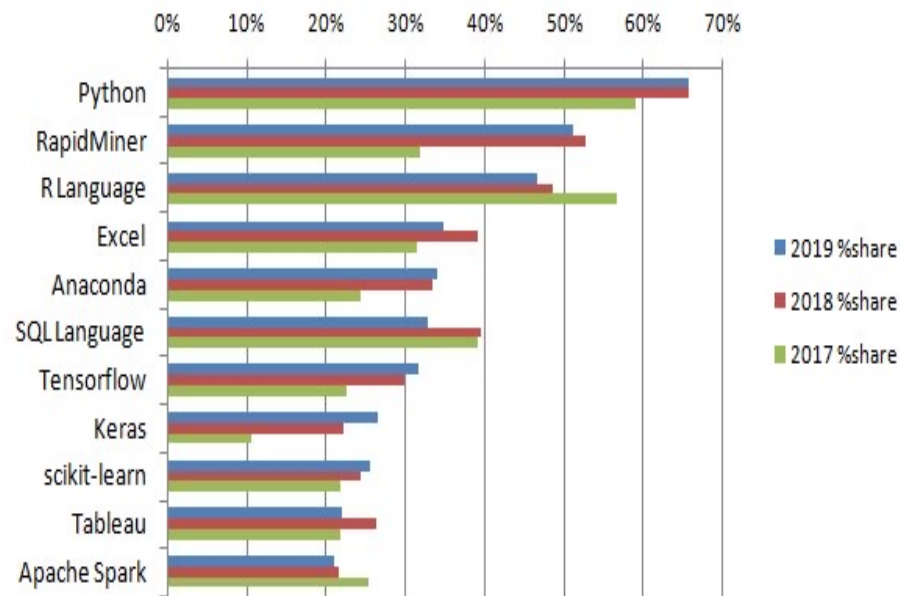
- A linguagem de programação Python foi criada em 1990 por Guido van Rossum no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda.
- Python é um software de código aberto (GPL), cuja especificação da linguagem é mantida pela *Python Software Foundation (PSF)*
- A implementação oficial é mantida pela PSF e escrita em C. Existem outras implementações, tais como:
 - IronPython: para .NET
 - Jython: para JVM
 - Pypy: em Python

Growth of major programming languages

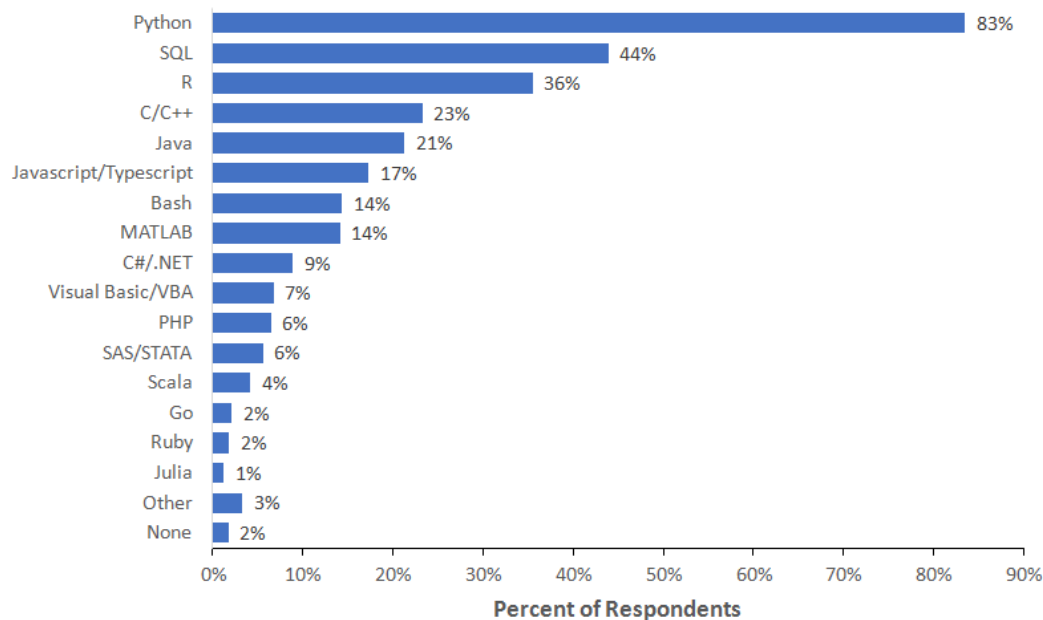
Based on Stack Overflow question views in World Bank high-income countries



Top Analytics, Data Science, Machine Learning Software 2017-2019, KDnuggets Poll



What programming language do you use on a regular basis?



Note: Data are from the 2018 Kaggle Machine Learning and Data Science Survey. You can learn more about the study here: <http://www.kaggle.com/kaggle/kaggle-survey-2018>. A total of 18827 respondents answered the question.

Python é uma linguagem de altíssimo nível, orientada a objetos, de tipagem dinâmica e forte, interpretada e interativa.

Linguagem de
Alto Nível

Orientada a
Objetos

Tipagem
Dinâmica e Forte

Interpretada

Interativa



Contém diversas estruturas de alto nível: listas, dicionários, data/hora, etc). É multiparadigma, ou seja, suporta programação funcional e modular além da orientação a objetos.

Python é uma linguagem de altíssimo nível, orientada a objetos, de tipagem dinâmica e forte, interpretada e interativa.

Linguagem de
Alto Nível

Orientada a
Objetos

Tipagem
Dinâmica e Forte

Interpretada

Interativa



Paradigma de
programação
composto
basicamente por 4
pilares: abstração,
encapsulamento,
herança e
polimorfismo

Python é uma linguagem de altíssimo nível, orientada a objetos, de tipagem dinâmica e forte, interpretada e interativa.

Linguagem de
Alto Nível

Orientada a
Objetos

Tipagem
Dinâmica e Forte

Interpretada

Interativa



Tipagem dinâmica: o
tipo de uma variável
é inferido pelo
interpretador em
tempo de execução

Tipagem forte: o
interpretador verifica
se as operações
são válidas e não faz
coerções
automáticas entre
tipos incompatíveis

Python é uma linguagem de altíssimo nível, orientada a objetos, de tipagem dinâmica e forte, interpretada e interativa.

Linguagem de
Alto Nível

Orientada a
Objetos

Tipagem
Dinâmica e Forte

Interpretada

Interativa



Precisa de um
interpretador para
que um programa
seja executado.

Não tem um
compilador que
transforma uma
linguagem
“humana” em
linguagem de
máquina

Python é uma linguagem de altíssimo nível, orientada a objetos, de tipagem dinâmica e forte, interpretada e interativa.

Linguagem de
Alto Nível

Orientada a
Objetos

Tipagem
Dinâmica e Forte

Interpretada

Interativa



Execução de
comandos via
shell

IDE:

- São **softwares** que integram várias ferramentas de desenvolvimento em um ambiente consistente
- Visual Studio Code, PyCharm, Spyder

Shell:

- São **ambientes interativos para execução de comandos**
- Ipython, Anaconda Prompt, Power Shell

Frameworks:

- São **coleções de componentes de software**
- Machine Learning:
 - ScikitLearn, PyTorch, Tensorflow, Keras
- Web:
 - Django, Web2Py
- Processamento Científico:
 - Numpy, Scipy

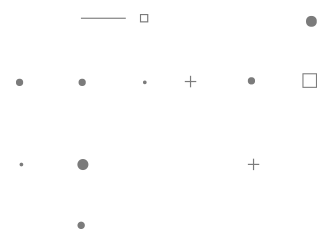
Anaconda



Jupyter Notebook e Gerenciamento de pacotes:

➤ Vou mostrar na prática

Sintaxe



Um programa feito em Python é constituído de linhas, que podem continuar nas linhas seguintes através do uso do caractere de barra invertida (\) ao final da linha, ou parênteses, colchetes ou chaves, em expressões que utilizam tais caracteres

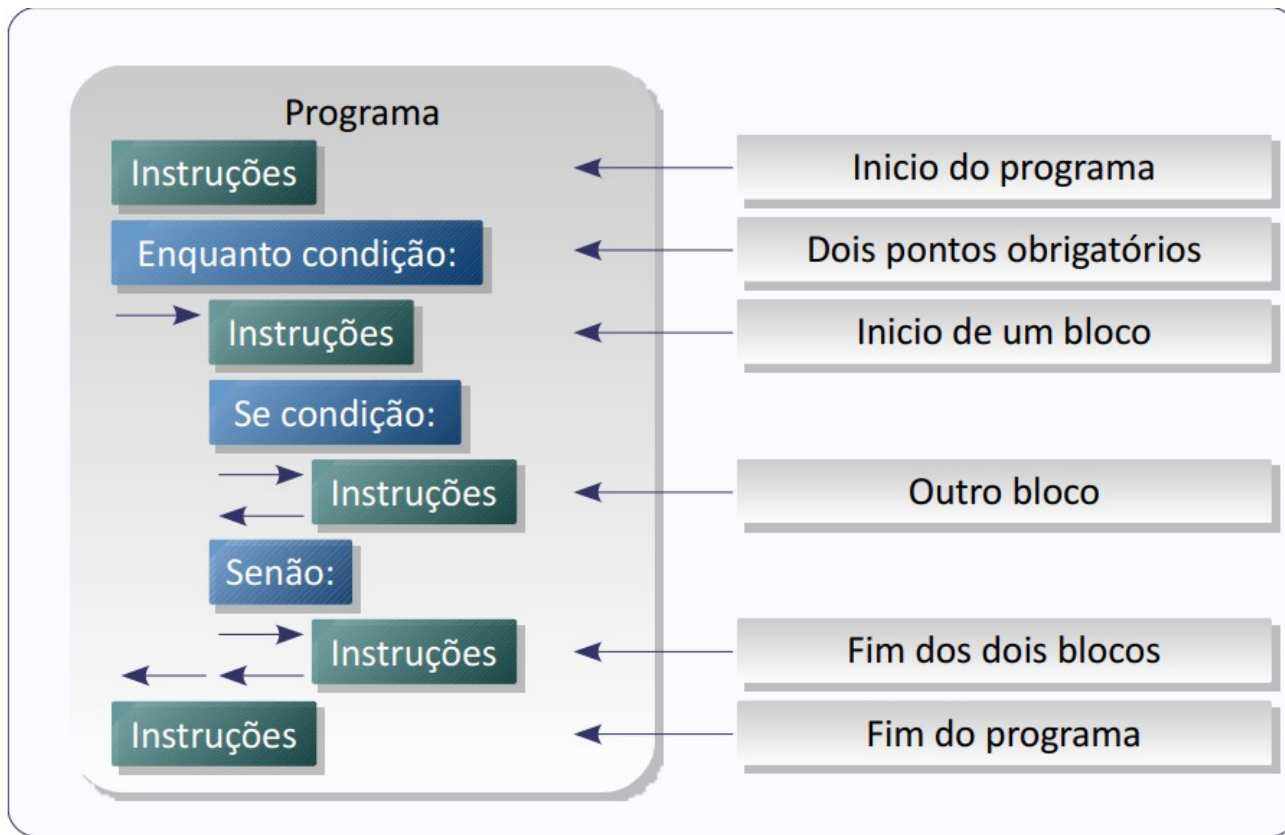
```
# -*- coding: latin1 -*-  
# Uma linha quebrada por contra-barra  
a = 7 * 3 + \  
5 / 2  
# Uma lista (quebrada por vírgula)  
b = ['a', 'b', 'c',  
     'd', 'e']  
# Uma chamada de função (quebrada por vírgula)  
c = range(1,  
11)  
# imprime todos na tela  
print a, b, c
```

O caractere # indica o início de um comentário. Qualquer texto depois de # será ignorado até o fim da linha.

Para comentários em mais de uma linha, use `""" """`. Este tipo de comentário é muito usado para docStrings

```
# comentário em apenas uma linha  
# para cada linha preciso adicionar o identificador de comentário #  
  
"""  
Este é um exemplo de bloco de comentário  
Comentário em mais de uma linha  
É mais usado para Docstrings: documentação e teste do código em python  
  
Para saber mais, acesse: https://www.python.org/dev/peps/pep-0257/  
"""
```

Em Python, blocos de código são delimitados pelo uso de endentação.



Vejam os um exemplo:

```
# Para i na lista 234, 654, 378, 798:  
for i in [234, 654, 378, 798]:  
# Se o resto da divisão por 3 for igual a zero:  
    if i % 3 == 0:  
        # Imprime...  
        print (i, '/ 3 =', i / 3)
```

É muito comum em um programa que certos conjuntos de instruções sejam executados de forma **condicional**, em casos como validar entradas de dados, por exemplo.

if <condição>:
 <bloco de código>
elif <condição>:
 <bloco de código>
elif <condição>:
 <bloco de código>
else:
 <bloco de código>

```
temp = int(input('Entre com a temperatura: '))  
if temp < 0:  
    print ('Congelando...')  
elif (0 <= temp <= 20):  
    print ('Frio')  
elif (21 <= temp <= 25):  
    print ('Normal')  
elif (26 <= temp <= 35):  
    print ('Quente')  
else:  
    print ('Muito quente!')
```

Se o bloco de código for composto de apenas uma linha, ele pode ser escrito após os dois pontos:

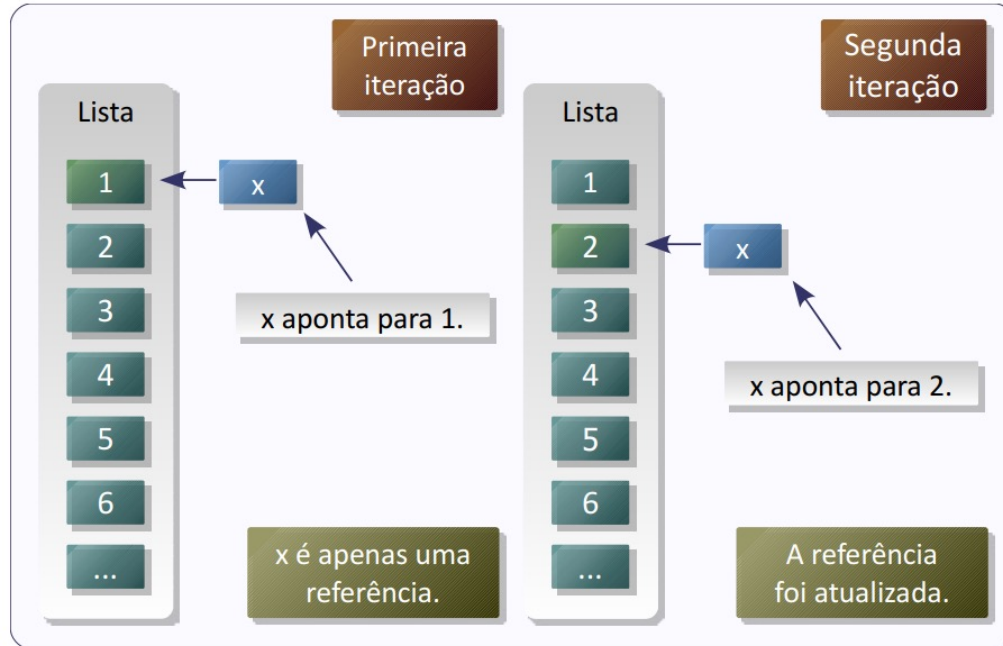
```
if temp < 0: print 'Congelando...'
```

Expressões ternárias também são aceitas:

- <variável> = <valor 1> if <condição> else <valor 2>

```
x = 'verdadeiro' if 1>0 else 'falso'
```

Laços (loops) são **estruturas de repetição**, geralmente usados para processar coleções de dados, tais como linhas de um arquivo ou registros de um banco de dados, que precisam ser processados por um mesmo bloco de código.



For Loop:

```
for <referência> in <sequência>:  
    <bloco de código>  
    continue  
    break  
else:  
    <bloco de código>
```

```
# Soma de 0 a 99  
s = 0  
for x in range(1, 100):  
    s = s + x  
print s
```

While Loop:

```
while <condição>:  
    <bloco de código>  
    continue  
    break  
else:  
    <bloco de código>
```

```
# Soma de 0 a 99  
s = 0  
x = 1  
  
while x < 100:  
    s = s + x  
    x = x + 1  
print s
```

```
parada = 10
inicio = 0

while parada > 0:
    print(parada)
    parada -= 1
    if parada == 3:
        break
    elif parada == 4:
        print('Volta pro inicio')
        continue
    else:
        pass
else:
    print("final")
```

Break: sai do loop

Continue: move o cursor para o top do loop

Pass: não altera nada, continua normal.

Uma instrução *match* pega uma expressão e compara seu valor com sucessivos padrões fornecidos como um ou mais blocos *case*. É superficialmente semelhantes à uma instrução *switch* em C, mas é mais semelhante ao *pattern matching* em Rust. Somente o primeiro padrão correspondente é executado.

```
#precisa python 3.10
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"
```

Resolver o exercício no notebook

Tipos de Dados



Variáveis em Python são criadas através de **atribuição**:

- O nome da variável deve começar com uma letra ou _
- Python é *case sensitive*

Os tipos mais comuns são implementados na forma de *builtins*, ou seja, sem necessidade de importar pacotes.

```
print ('int(3.14) =', int(3.14)) # real para inteiro
print ('float(5) =', float(5)) # inteiro para real
print ('5.0 / 2 + 3 = ', 5.0 / 2 + 3) #Divisão que resulta em real
print ('5/4 ou 5.0/4.0 = ', 5//4) #divisão sem resto
c = 4 + 3j #número complexo
print ('c =', c)
print ('Parte real:', c.real)
print ('Parte imaginária:', c.imag)
print ('Conjugado:', c.conjugate())
```

```
int(3.14) = 3
float(5) = 5.0
5.0 / 2 + 3 = 5.5
5/4 ou 5.0/4.0 = 1
c = (4+3j)
Parte real: 4.0
Parte imaginária: 3.0
Conjugado: (4-3j)
```

| | | |
|----|---|--|
| + | Soma | $1 + 2 = 3$ |
| - | Subtração | $2 - 1 = 1$ |
| * | Multiplicação | $2 * 2 = 4$ |
| / | Divisão | $5 / 4 = 1.25$ |
| // | Divisão inteira (o resultado é truncado para o inteiro imediatamente inferior) | $5 // 4 = 1$ $5.0 // 4.0 = 1.0$ |
| % | Módulo (retorna o resto da divisão) | $5 \% 4 = 1$ |
| ** | Potência | $10 ** 2 = 100$ $100 ** 0.5 = 10.0$ |

| | | |
|----|--|-----------------------|
| == | Se os valores dos dois operadores forem igual, a condição é verdadeira | (1 == 2) is not true. |
| != | Se os valores dos dois operadores não forem igual, a condição é verdadeira | (1 != 2) is true. |
| > | Se o valor do operador da esquerda for maior que o da direita, a condição é verdadeira | (1 > 2) is not true. |
| < | Se o valor do operador da direita for maior que o operador da esquerda, a condição é verdadeira | (1 < 2) is true. |
| >= | Se o valor do operador da esquerda for maior ou igual que o da direita, a condição é verdadeira | (1 >= 2) is not true. |
| <= | Se o valor do operador da direita for maior ou igual que o operador da esquerda, a condição é verdadeira | (1 <= 2) is true. |

| | | |
|-----------|---|--|
| = | Atribui ao operando da esquerda o valor do operando da direita | $a = 1$ |
| += | Atribui ao operando da esquerda a soma do operando da esquerda com o da direita | $a += b$ é equivalente a $a = a + b$ |
| -= | Atribui ao operando da esquerda a subtração do operando da esquerda com o da direita | $a -= b$ é equivalente a $a = a - b$ |
| *= | Atribui ao operando da esquerda a multiplicação do operando da esquerda com o da direita | $a *= b$ é equivalente a $a = a * b$ |
| /= ou //= | Atribui ao operando da esquerda a divisão ou divisão inteira do operando da esquerda com o da direita | $a /= b$ é equivalente a $a = a / b$ |
| %= | Atribui ao operando da esquerda o módulo do operando da esquerda com o da direita | $a \% = b$ é equivalente a $a = a \% b$ |
| **= | Atribui ao operando da esquerda a potência do operando da esquerda com o da direita | $a ** = b$ é equivalente a $a = a ** b$ |

Strings são **variáveis imutáveis**, ou seja, não é possível adicionar, remover ou mesmo modificar algum caractere de uma string.

```
s = 'Camel'
print ('The ' + s + ' run away!')
print ('tamanho de %s => %d' % (s, len(s))) # interpolação
for ch in s: print (ch)
```

```
The Camel run away!
tamanho de Camel => 5
C
a
m
e
l
```

Símbolos usados na interpolação:

- %s: *string*.
- %d: inteiro.
- %o: octal.
- %x: hexadecimal.
- %f: real.
- %e: real exponencial.
- %: sinal de percentagem.

Exemplos de interpolação:

```
print ('Agora são %02d:%02d.' % (16, 30))  
print ('Percentagem: %.1f%%, Exponencial:%.2e' % (5.333, 0.00314)) #casa decimal e formatação  
print ('Decimal: %d, Octal: %o, Hexadecimal: %x' % (10, 10, 10)) #bases numéricas
```

Agora são 16:30.

Percentagem: 5.3%, Exponencial:3.14e-03

Decimal: 10, Octal: 12, Hexadecimal: a

A função *format()* é outra forma de interpolação:

```
musicos = [('Page', 'guitarrista', 'Led Zeppelin'),  
            ('Fripp', 'guitarrista', 'King Crimson')]  
  
msg = '{0} é {1} do {2}'  
  
for nome, funcao, banda in musicos:  
    print(msg.format(nome, funcao, banda))
```

```
Page é guitarrista do Led Zeppelin  
Fripp é guitarrista do King Crimson
```

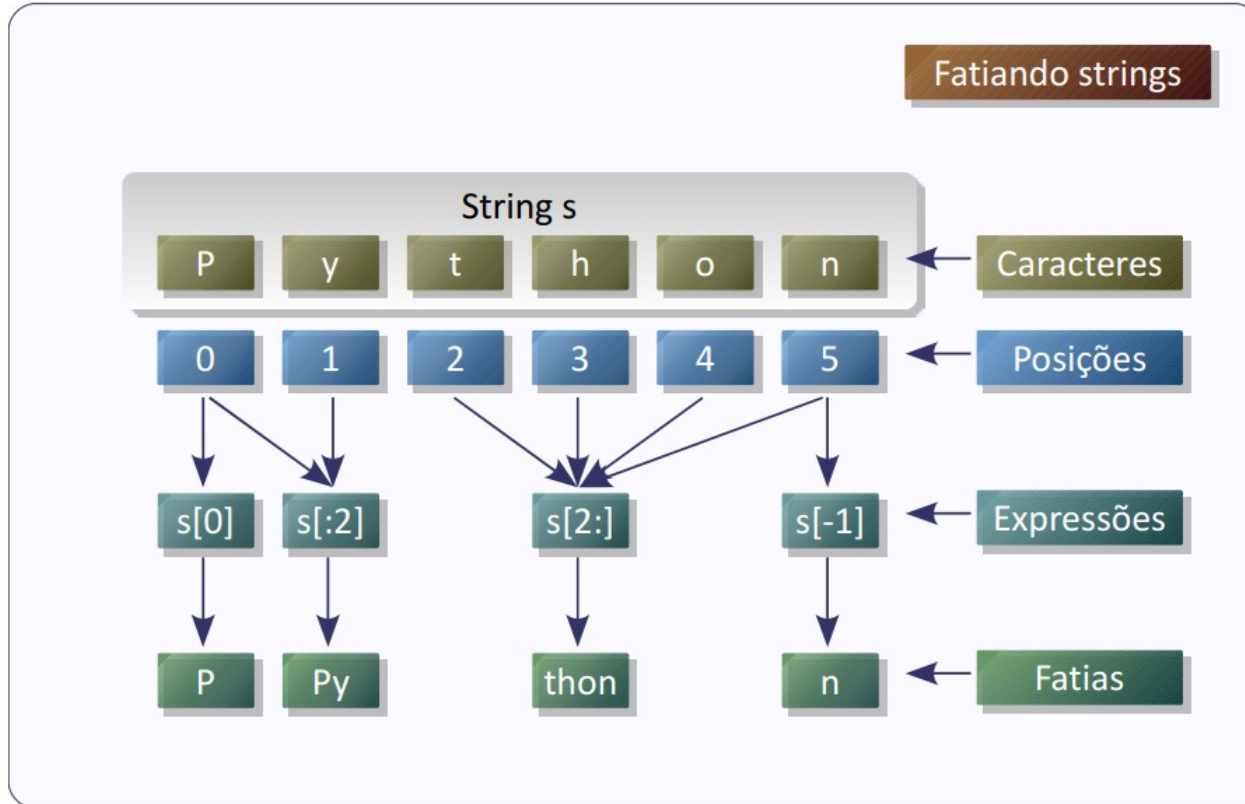
Raw strings:

```
# Caracteres antecipados por \ são interpretados como caracteres especiais;  
# se você quer evitar isso, use raw strings, adicionando um r antes da primeira aspas  
  
print ('C:\some\name')  
print (r'C:\some\name')
```

Python 3.6 introduziu uma nova feature denominada *f-strings*, que torna o processo de criação de strings formatadas ainda mais conveniente. Para criar uma *f-string*, escreva o caracter “f” imediatamente antes de uma string literal. Dentro da string, mantenha as expressões entre chaves para substituir o valor da expressão na string formatada, como o exemplo a seguir:

```
quantidade = 10
cambio = 5.15
moeda = "Reais"
result = f"{quantidade} {moeda} valem US${quantidade / cambio}"
#caso queira limitar a quantidade de números após a vírgula
#result = f"{quantidade} {moeda} valem US${quantidade / cambio:.2f}"
result
```

É possível fatiar uma string (processo conhecido como *slicing*):



Listas são **coleções heterogêneas de objetos**, que podem ser de qualquer tipo, inclusive outras listas.

As listas no Python são **mutáveis**, podendo ser alteradas a qualquer momento. Listas podem ser fatiadas da mesma forma que as *strings*, mas como as listas são mutáveis, é possível fazer atribuições a itens da lista.

Sintaxe: `lista = [1,2,3]`

```
# listas são objetos heterogêneos, ou seja,  
# podem conter elementos de qualquer tipo,  
# inclusive outras listas  
  
integer_list = [1, 2, 3]  
heterogeneous_list = ["string", 0.1, True]  
list_of_lists = [ integer_list, heterogeneous_list, [] ]  
  
print(integer_list)  
print(heterogeneous_list)  
print(list_of_lists)
```

```
bandas = ['Queen', 'Led Zeppelin', 'Kansas', 'Europe', 'Guns N Roses']

for banda in bandas:
    print (banda)

bandas.append('Bon Jovi') #adiciona ao fim da lista
bandas.remove('Europe') #remove
bandas.sort() #ordena

print (bandas)
```

Queen

Led Zeppelin

Kansas

Europe

Guns N Roses

['Bon Jovi', 'Guns N Roses', 'Kansas', 'Led Zeppelin', 'Queen']

```
for i, banda in enumerate(bandas):  
    print (i + 1, '=>', banda)  
  
#filas e pilhas  
while bandas:  
    print ('Saiu', bandas.pop(0), ', faltam', len(bandas))
```

```
1 => Bon Jovi  
2 => Guns N Roses  
3 => Kansas  
4 => Led Zeppelin  
5 => Queen  
Saiu Bon Jovi , faltam 4  
Saiu Guns N Roses , faltam 3  
Saiu Kansas , faltam 2  
Saiu Led Zeppelin , faltam 1  
Saiu Queen , faltam 0
```

A função `enumerate()` retorna uma tupla de dois elementos a cada iteração: um número sequencial e um item da sequência correspondente.

Semelhantes às listas, porém são **imutáveis**: não se pode acrescentar, apagar ou fazer atribuições aos itens.

Sintaxe: `tupla = (a,b,c)`

Particularidade: tupla com apenas um elemento

`T1 = (1,)`

Conversões

Lista em tupla: `tupla = tuple(lista)`

Tupla em lista: `lista = list(tupla)`

Tuplas são uma forma conveniente de retornar múltiplos valores de uma função:

```
def soma_e_produto(x, y):  
    return (x + y), (x * y)  
sp = soma_e_produto(2, 3) # igual (5, 6)  
s, p = soma_e_produto(5, 10) # s é 15, p é 50  
print(sp)  
print(s)  
print(p)
```

Um dicionário é uma **lista de associações compostas por uma chave única e estruturas correspondentes**. Dicionários são mutáveis, tais como as listas.

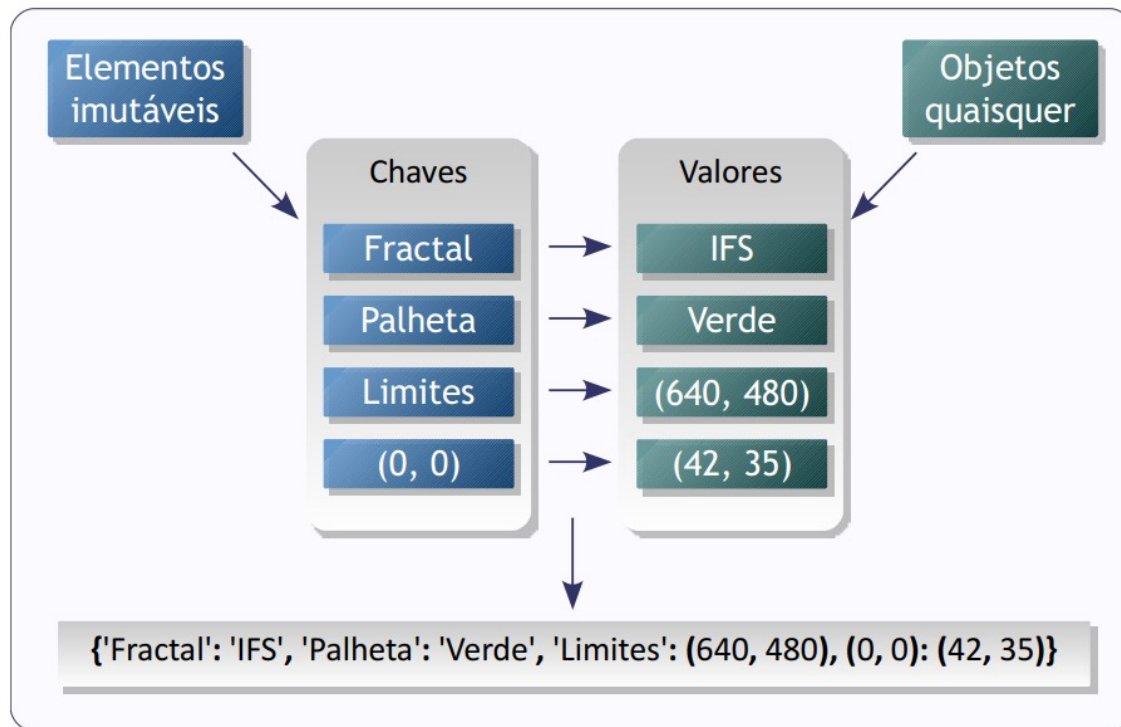
A chave precisa ser de um tipo imutável, já os itens podem ser de qualquer tipo.

- Não há garantia de ordenação de chaves

Sintaxe:

```
dicionario = {'a': a, 'b': b, ..., 'z': z}
```

Estrutura:




```
carros = {'marca': 'Ford', 'modelo': 'Mustang', 'Ano': 1964} #cria um dicionário
print(carros['modelo']) #imprime o valor da chave 'modelo'
print(carros.get('Ano')) #imprime usando método get
carros['Ano'] = 2018

for x,y in carros.items():#imprimindo keys e values
    print(x,y)

carros['cor'] = 'preto' #inserindo nova key e value

for x,y in carros.items():#repetindo após inserção
    print(x,y)

del carros['Ano'] #apagando key e value -
#pode ser usado os métodos pop e popitem - testar
print(carros) #imprime dicionário

#Apagar dicionário
#del carros
#print(carros)
#esvaziar dicionário
carros.clear()
print(carros)

#criando a partir de um construtor
carros = dict(marca='Ford', modelo='Mustang', Ano=1964)
print(carros)
```

Mustang

1964

marca Ford

modelo Mustang

Ano 2018

marca Ford

modelo Mustang

Ano 2018

cor preto

```
{'marca': 'Ford', 'modelo': 'Mustang', 'cor': 'preto'}
```

```
{}
```

```
{'marca': 'Ford', 'modelo': 'Mustang', 'Ano': 1964}
```

Imagine que você precise contar as palavras num documento. Uma abordagem óbvia é criar um dicionário no qual as chaves são as palavras e os valores são as contagens. Conforme você verifica cada palavra, incrementa sua contagem se ela já estiver no dicionário ou a adiciona caso ela não esteja:

```
word_counts = {}  
for word in document.split(' '):  
    if word in word_counts:  
        word_counts[word] += 1  
    else:  
        word_counts[word] = 1
```

Entretanto, temos opções mais simples. Podemos usar, então, defaultdicts para resolver esse problema de modo mais elegante. Um defaultdict é como um dicionário regular, entretanto não possui uma chave, mas cria um item default baseado no argumento passado no construtor. Por exemplo, se for um int, vai retornar um objeto inteiro com valor 0; se for uma lista, retornará uma lista vazia.

```
from collections import defaultdict

word_counts = defaultdict(int) # int() produces 0
for word in document.split(' '):
    word_counts[word] += 1
```

Um Counter converte uma sequência de valores num objeto semelhante a um defaultdict, mapeando chaves para valores.

```
from collections import Counter
c = Counter([0, 1, 2, 0])
c
```

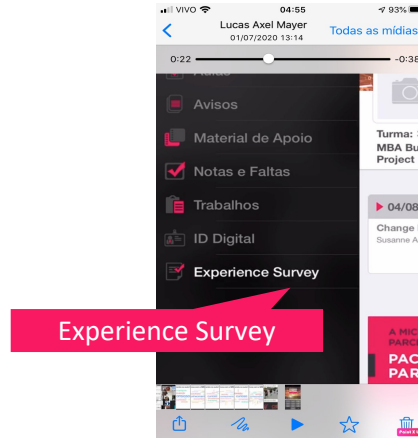
Isso nos permite resolver o problema do *word_counts* de maneira muito simples:

```
word_counts_counter = Counter(document.split(' '))
word_counts_counter
```

Resolver o exercício no notebook

O que você achou da aula de hoje?


Entrar no aplicativo FIAPP, e no menu clicar em Experience Survey



Ou pelo link: <https://fiap.me/Pesquisa-MBA>

Obrigado!

profdheny.fernandes@fiap.com.br

 /dhenyfernandes

FIAP MBA⁺

Copyright © 2018 | Professor Dheny R. Fernandes

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.

FIAP