Linguagem de Programação

- ## Cw1.1

  Python é uma linguagem de programação orientada a objetos.

  Uma variável é um espaço alocado na memória RAM. O tipo de variável é identificado no momento que se atribui um valor a ela.

  Os códigos a seguir exibem o resultado para três objetos: **int**, **str** e **float** (tipos de variáveis), sem declaração do tipo.

```
print('olá')
# string
print(10)
# inteiro
print(3.141592)
# float
```


```
>>> valor = 10
>>> type(valor)
<class 'int'>
```


```
>>> valor = 'nome'
>>> type(valor)
<class 'str>
```


```
>>> valor = 10.5
>>> type(valor)
<class 'float'>>
```

- ## Cw1.2 – Estruturas Condicionais de repetição em Python

  Existem o if; else; elif. Colocar dois-pontos (:) ao final da condição inicial.

Praticando

Imagine que você deseja acessar pela primeira vez algum sistema protegido por senha. Nesse exercício vamos implementar um simples verificador de senha do usuário e para isso utilizaremos as estruturas condicionais para permitir o login de um usuário. Vamos aplicar apenas as estruturas ***if-else***.

Observe a sintaxe para esse comando:

```
login = input("Digite o seu login:")
senha = input("Digite sua senha: ")
if login == "userpy" and senha == "teste123":
    print("Bem-vindo userpy01")
else:
    print("Login falhou, verifique sua senha e tente novamente")
```

Fonte: elaborada pela autora.

A instrução FOR: for **variável** in **sequencia: (lembrar dois-pontos)**

A função **range() = list (range(10)) #Gera uma sequência de 0 a 9.**

O loop **for** é muito utilizado com o **range.** Facilita a iteração dos valores sem a necessidade de escrever código para alterar o valor. Ex:

```
contagem = 0
for contagem in range(1,10):
    print(contagem).
```

While: Quando queremos solicitar e testar se o número digitalo pelo usuário é par ou ímpar. Quando ele digitar zero, o programa se encerra!
A estrutura **while** só é interrompida quando a condição passa a ser falsa.

```
contagem = 0
    while (contagem < 10):
        print(contagem).
        Contagem = contagem + 1
```

Estruturas lógicas and, or, not.

and = retorna um valor verdadeiro somente se as duas expressões forem verdadeiras.

or = retorna um valor falso somente se as duas expressões forem falsas

not = esse operador muda o valor de seu arumento. Se o argumento for verdadeiro, a operação o transformará em falso e vice-versa.

- **Cw1.3 – Funções em Python**

Funções *built-in* é um objeto que está integrado ao núcleo do interpretador Python. É como se fosse funções já prontas para serem utilizadas no Python.

Algumas destas funções são:

print() = usada para imprimir um valor na tela.
enumarate() = usada para retornar a posição de um valor em uma sequência.
input() = usada para capturar um valor digitado no teclado.
int() e float() = usada para converter um valor no tipo inteiro ou float.
type() = usada para saber qual é o tipo de um objeto.

Podemos também criar funções. Que é utilizada para criar ***def.***

def nome_funcao ():

bloco de comandos que vai ser utilizado na função

Quando criar um função com argumentos¿ Tirar dúvida em live.

Quando utilizar a função anômias¿ lambda (não é necessário utilizar o ***def***.

Linguagem de Programação

- **Cw2.1 – Listas, tuplas , set e dicionário.**

Listas – estrutura de dados mutável. Ou seja, novos valores podem ser adicionados ou removidos.

A ultima posição é n – 1

Ex: vogais = [‘a’, ‘e’, ‘i’, ‘o’, ‘u’]

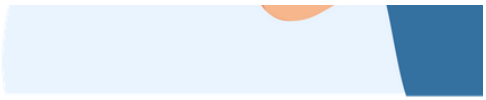
A lista pode ser criada sem nenhum elemento e feita posteriormente:

Para acessar o valor guardado em Uma lista, basta indicar o nome da Variável e a posição do elemento.
Ex: vogais[2]

```
vogais = []
vogais.append('a')
vogais.append('e')
vogais.append('i')
vogais.append('o')
vogais.append('u')
```

List Comprehension – também chamada de **listcomp**

Sua sintaxe básica é:
[item **for** item **in** lista]



Fonte: Shutterstock.

Os comandos **for-in** são obrigatórios.

As variáveis **item** e **lista** dependem do nome dado no programa. Veja um exemplo de sintaxe utilizando a listcomp:

```
[2*x for x in range(10)]
```

Tuplas – estrutura de dados de objetos tipo sequencia imutáveis.

Uma das maneiras de criar uma tupla é colocando valores entre **parênteses**. Ao contrario da lista, uma tupla não permite a inserção posterior de dados, mas os dados podem ser acessados pela sua posição na sequencia.

Ex: vogais = (‘a’, ‘e’, ‘i’, ‘o’, ‘u’)

Sets – estrutura para operações matemáticas de conjuntos, tais como: união, intersecção, diferença e etc. Uma das maneiras de se criar um objeto **set** é colocnado os valores entre **chaves**. Um **set** permite a inserção de valores posteriores a sua criação com a função **add()**, mas não permite acessar valores pela sua posição.

Ex: vogais = {‘a’, ‘e’, ‘i’, ‘o’, ‘u’}

Dicionários – estrutura de dados que possuem um mapeamento entre uma chave e um valor são considerados objetos do tipo **mapping.**

Em Python, uma das maneiras de criar um objeto do tipo dicionário é colocando as chaves e os valores entre estas, conforme código a seguir:

```
cadastro = {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}
```

Para acessar um valor em um dicionário, basta digitar:

```
nome_dicionario[chave]
```

E, para atribuir um novo valor, use:

```
nome_dicionario[chave] = novo_valor
```

- **Cw2.2 – Algoritmos de busca**

Busca **Sequencial –** É necessário percorrer todos os elementos da lista até encontrar o elemento procurado.

Algoritmo de busca binária – A diferença entre algoritmo de busca linear e algoritmo de busca binária é que neste os valores precisam estar ordenados.

A lógica é a seguinte:

1. Encontra o item no meio da sequência.
2. Se o valor procurado for igual ao item do meio, a busca encerra.
3. Se não, verifica se o valor buscado é maior ou menor que o valor central.
4. Se for maior, então a busca acontecerá na metade superior da sequência (a inferior é descartada), se não for maior, a busca acontecerá na metade inferior da sequência (a superior é descartada).
5. Repete os passos: 1, 2, 3, 4.

Atenção: Veja que o algoritmo, ao encontrar o valor central de uma sequência, a divide em duas partes, o que justifica o nome de busca binária.

- **Cw2.3 – Algoritmos de ordenação**

Classificação dos algoritmos –

Complexidade **O(N2).** Nesse grupo estão os algoritmos **selection sort, bubble sort e insertion sort.** São algoritmos lentos para ordenar em grandes listas porém são intuitivos de entender e possuem fácil codagem.

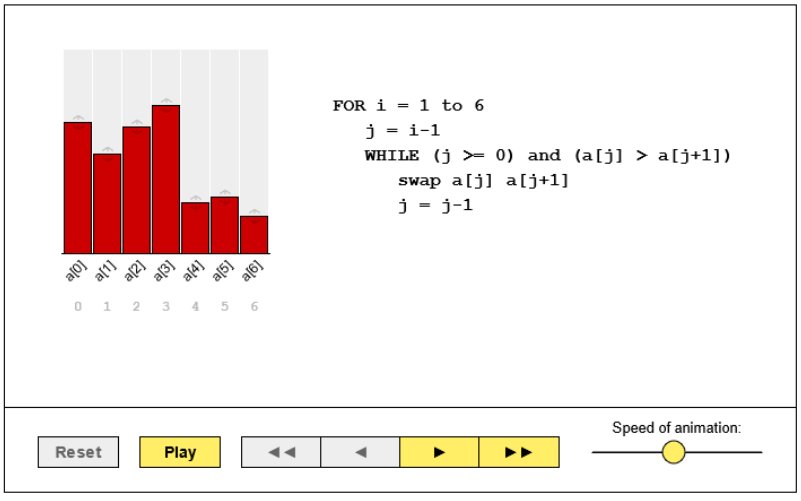
Complexidade **O(N log N)**. Neste grupo, vamos conhecer **merge sort e quick sort.** Possuem performance superior, porém são um pouco mais complexos de serem implementados.

Algoritmo de Ordenação por inserção (*Insertion Sort*)

O algoritmo de ordenação por inserção é iniciado a partir do segundo valor no vetor, pois o primeiro elemento será uma referência para a ordenação. Ou seja, o segundo elemento do vetor será comparado com o primeiro.

O vetor é percorrido da esquerda para a direita. Nesse caminho, compara-se sempre o elemento da direita com os elementos à esquerda de modo que os elementos mais à esquerda sejam organizados e ordenados. O algoritmo de ordenação por inserção tem funcionamento similar ao das pessoas que ordenam cartas em um jogo de baralho.

A animação a seguir mostra o funcionamento do algoritmo de ordenação por inserção.



Fonte: https://anim.ide.sk/sorting_algorithms_1.php

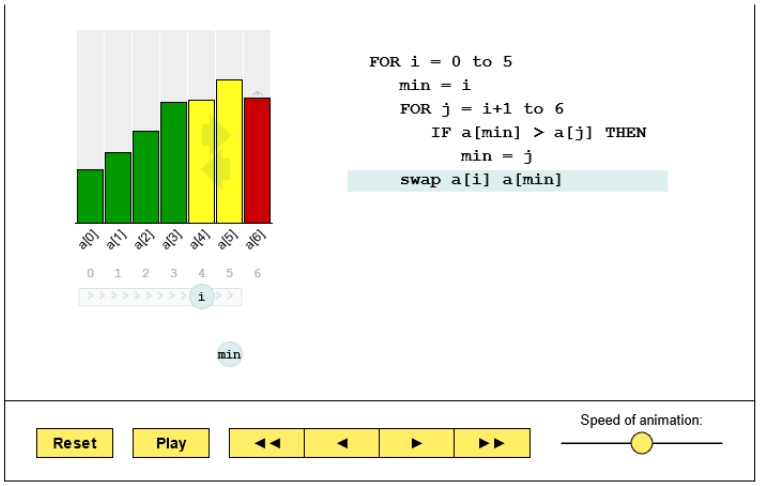
```
1  def executar_insertion_sort(lista):
2      n = len(lista)
3      for i in range(1, n):
4          valor_inserir = lista[i]
5          j = i - 1
6
7          while j >= 0 and lista[j] > v
8              lista[j + 1] = lista[j]
9              j -= 1
10         lista[j + 1] = valor_inserir
11
12     return lista
13
14
15 lista = [10, 8, 7, 3, 2, 1]
16 executar_insertion_sort(lista)
```

Algoritmo de ordenação por seleção (*selection sort*)

Seleção pelo valor mínimo

O princípio de funcionamento deste algoritmo é transferir o menor valor do vetor para a primeira posição e, em seguida, o segundo menor valor para a segunda posição, e assim sucessivamente, para os n-1 elementos até os últimos dois elementos.

A animação a seguir mostra o funcionamento do algoritmo de ordenação por seleção.



Fonte: http://anim.ide.sk/sorting_algorithms_1.php

Python 3.6

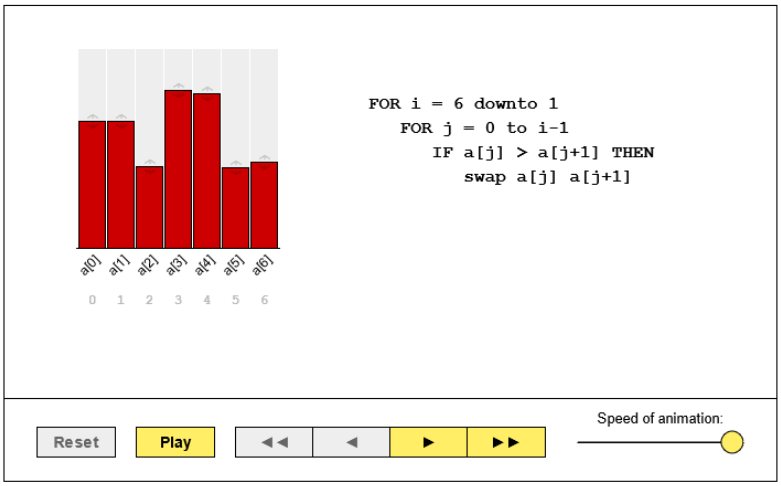
```
1  def executar_selection_sort(lista):
2      n = len(lista)
3
4      for i in range(0, n-1):
5          min = i
6          for j in range(i+1, n):
7              if lista[j] < lista[min]:
8                  min = j
9          lista[i], lista[min] = lista[
10     return lista
11
12
13 lista = [10, 8, 7, 3, 2, 1]
14 executar_selection_sort(lista)
```

Algoritmo de ordenação por bolha (*bubble sort*)

O algoritmo de ordenação por bolha (ou flutuação) consiste naquele baseado em comparações, em que se percorre vetor ou lista múltiplas vezes. A cada passagem, os pares de elementos adjacentes do vetor são comparados e, depois disso, são trocados se não estiverem ordenados. Esse processo segue para todos os elementos não ordenados da lista.

É importante destacar que esse tipo de algoritmo não é indicado para uma grande quantidade de dados.

A animação a seguir mostra o funcionamento do algoritmo de ordenação por bolha (flutuação).




```
1  def executar_bubble_sort(lista):
2      n = len(lista)
3      for i in range(n-1):
4          for j in range(n-1):
5              if lista[j] > lista[j + 1
6                  lista[j], lista[j + 1
7      return lista
8
9  lista = [10, 8, 7, 3, 2, 1]
10 executar_bubble_sort(lista)
```

Linguagem de Programação

- # Cw3.1 – Classes e métodos em Python

Definições importantes

Antes de aprendermos como criar uma classe em Python vamos conhecer os conceitos de objeto, classe e instância.

» Objetos

São os componentes de um programa OO. Um programa que usa a tecnologia OO é basicamente uma coleção de objetos.

» Classe

Uma **classe** é um modelo para um objeto.
Segundo a *Python Software Fundation* (PSF, 2020a), podemos considerar uma classe como uma forma de organizar os dados (de um objeto) e seus comportamentos.

» Instância

Entende-se por **instância** a existência física, em memória, do objeto.

Como criar uma classe em Python

Atributos

Para criar uma classe em Python é necessária a sintaxe a seguir. Utiliza-se a palavra reservada "class" para indicar a criação de uma classe, seguida do nome e de dois pontos. No bloco indentado devem ser implementados os atributos e métodos da classe.

```
class ClassName:
     < statement-1 >
    .
    .
    .
     < statement-N >
```

Exemplo de criação de uma classe

```
class PrimeiraClasse:
     nome = None

     def imprimir_mensagem(self):
         print("Olá seja bem vindo!")
```

Após criada uma classe, os objetos podem ser instanciados, sendo importante lembrar que uma classe determina um tipo de estrutura de dados. Os atributos e os métodos de uma classe podem ser acessados pelo objeto, colocando o nome deste seguido de ponto; por exemplo: `objeto.atributo`.
A seguir apresentamos um exemplo.

```
objeto1 = PrimeiraClasse()
objeto1.nome = "Aluno 1"

print(objeto1.nome)
objeto1.imprimir_mensagem()
```

Os dados armazenados em um objeto representam o estado do objeto. Na terminologia de programação OO, esses dados são chamados de **atributos**. Os atributos contêm as informações que diferenciam os vários objetos – neste caso, os funcionários.

Construtor da classe – __init__()

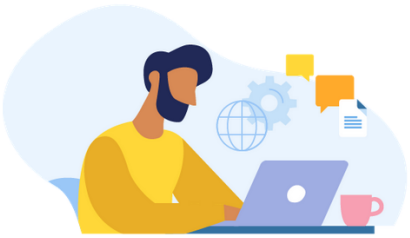
Ao instanciar um novo objeto, é possível determinar um estado inicial para variáveis de instâncias (atributos) por meio do método construtor da classe.

Em Python, o método construtor é chamado de *__init__()* e deve ser usado conforme o código a seguir. Na classe *FuncionarioTecnico*, o atributo status, que é uma variável de instância, recebe o valor no momento da criação do objeto, pois está no construtor.

```
class FuncionarioTecnico:
def __init__(sef, status):
    self.status = status

nivel = 'Técnico'
func1 = FuncionarioTecnico('Ativo')
func2 = FuncionarioTecnico('Licença Mestrado')

print(func1.nivel)
print(func2.nivel)
print(func1.status)
print(func2.status)
```



Fonte: Shutterstock.

Todo método em uma classe deve receber como primeiro parâmetro uma variável que indica a referência à classe – por convenção, adota-se o parâmetro **self**. O parâmetro **self** será usado para acessar os atributos e métodos dentro da própria classe.

Toda variável de instância possui o prefixo **self**, pois é dessa forma que é identificado que o atributo faz parte de um objeto específico.
Para se utilizar um método, dentro da classe, também é necessário utilizar o prefixo self.

Métodos

O comportamento de um objeto representa o que o objeto pode fazer. Nas linguagens procedurais, o comportamento é definido por:

- Procedimentos
- Funções
- Sub-rotinas.

Na terminologia de programação OO, esses comportamentos estão contidos nos **métodos**, e você invoca um método enviando uma mensagem para ele.

- # Cw3.2 – Bibliotecas e módulos em Python.

Organização em módulos

Uma opção para organizar o código é implementar funções, contexto em que cada bloco passa a ser responsável por uma determinada funcionalidade. Outra forma é utilizar a orientação a objetos e criar classes que encapsulam características e comportamentos de um determinado objeto. Conseguimos utilizar ambas as técnicas para melhorar nosso código, mas, ainda assim, estamos falando de toda a solução agrupada em um arquivo Python (.py).

Atenção

Considerando a necessidade de implementar uma solução, o mundo ideal é:

Modular uma solução ou seja fazer a separação em funções ou classes e ainda realizar a separação em vários arquivos .py (PSF, 2020b).

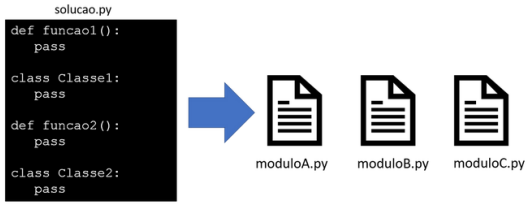
Segundo a documentação oficial do Python, é preferível implementar, em um arquivo separado, uma funcionalidade que você possa reutilizar, criando assim um módulo.

"Um módulo é um arquivo contendo definições e instruções Python. O nome do arquivo é o nome do módulo acrescido do sufixo .py." (PSF, 2020b, [s.p.]).

Criação de módulos

Veja a seguir essa ideia, com base na qual uma solução original implementada em um único arquivo .py é transformada em três módulos que, inclusive, podem ser reaproveitados, conforme aprenderemos.

Contínuo *versus* módulo



Fonte: elaborada pela autora.

Bibliotecas

Falamos em módulo, mas em Python se ouve muito o termo biblioteca. O que será que eles têm em comum?

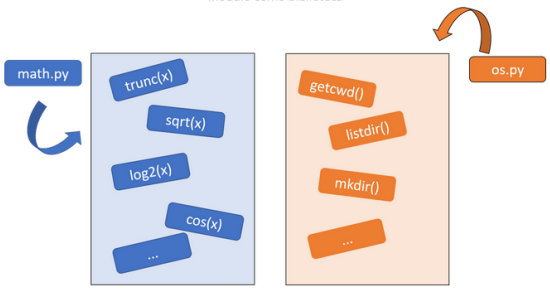
Na verdade, um módulo pode ser uma biblioteca de códigos! Veja a seguir que temos o módulo ***math***, que possui diversas funções matemáticas, e o módulo ***os***, que possui funções de sistema operacional, como capturar o caminho (getcwd), listar um diretório (listdir), criar uma nova pasta (mkdir), dentre inúmeras outras.

Esses módulos são bibliotecas de funções pertinentes a um determinado assunto (matemática e sistema operacional), as quais possibilitam a reutilização de código de uma forma elegante e eficiente.

Blibliotecas

math.py

Módulo como biblioteca



Fonte: elaborada pela autora.

Como utilizar um módulo

Para utilizar um módulo, é preciso importá-lo para o arquivo. Essa importação pode ser feita de maneiras distintas:

» `1. import moduloXX`

`1.2 import moduloXX as apelido`

Desta forma todas as funcionalidades de um módulo são carregadas na memória.

A diferença entre elas é que, na primeira, usamos o nome do módulo e, na segunda, atribuímos um apelido (as = alias) ao módulo.

» `2. from moduloXX import itemA, itemB`

Nesta forma de importação, somente funcionalidades específicas de um módulo são carregadas na memória. A forma de importação também determina a sintaxe para utilizar a funcionalidade. Com podemos observar nos exemplo a seguir:

Exemplo 1: Importando todas as funcionalidades da biblioteca

```
import math

math.sqrt(25)
math.log2(1024)
math.cos(45)
```

< Importando toda bliblioteca :
import **math**

Exemplo 2: Importando todas as funcionalidades da biblioteca com alias

```
import math as m

math.sqrt(25)
math.log2(1024)
math.cos(45)
```

< importando toda bliblioteca e definindo ‘m’ como um apelido\variavel

Exemplo 3: Importando funcionalidades específicas de uma biblioteca

```
from math import sqrt, log2, cos

sqrt(25)
log2(1024)
cos(45)
```

< importando apenas algumas funcionalidades da Blilioteca.

Classificação dos módulos (bibliotecas)

Podemos classificar os módulos (bibliotecas) em três categorias:

» **Módulos *built-in***
São embutidos no interpretador. Ao instalar o interpretador Python, também é feita a instalação de uma biblioteca de módulos, que pode variar um de sistema operacional para outro.

» **Módulos de terceiros**
São criados por terceiros e disponibilizados via PyPI.
PyPI é a abreviação para *Python Package Index*, que é um repositório para programas Python. Programadores autônomos e empresas podem criar uma solução em Python e disponibilizá-la em forma de biblioteca no repositório PyPI. Dessa forma, todos usufruem e contribuem para o crescimento da linguagem.

» **Módulos próprios**
São criados pelo desenvolvedor. Cada módulo pode importar outros módulos, tanto os pertencentes ao mesmo projeto, quanto os built-in ou de terceiros.

Módeulos built-in = Já estão instalado blibliotecas e funções dentro do Py.
Módulos de terceiros = São criações de terceiros
Módulos próprios = são próprios kkk

Cw3.3 – Aplicação de banco de dados com Python

A linguagem SQL

Para se comunicar com um banco de dados relacional, existe uma linguagem específica conhecida como SQL, que significa Structured Query Language ou, traduzindo, linguagem de consulta estruturada. As instruções da linguagem SQL são divididas em três grupos: DDL; DML; DCL (ROCKOFF, 2016).

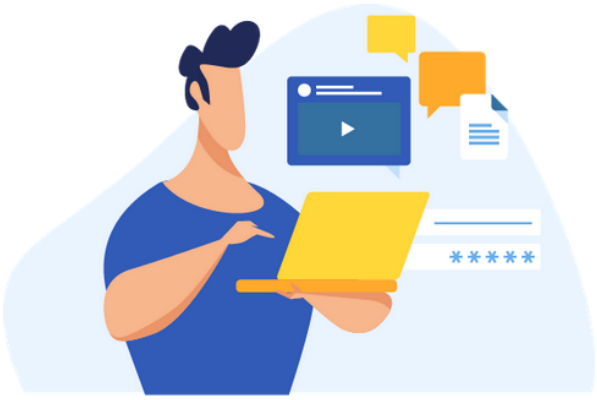
DDL é um acrônimo para data *definition language* (linguagem de definição de dados). Fazem parte desse grupo as instruções destinadas a **criar**, **deletar** e **modificar** banco de dados e tabelas. Nesse módulo, vão aparecer comandos como CREATE, ALTER e DROP.

DML é um acrônimo para data *manipulation language* (linguagem de manipulação de dados). Fazem parte deste grupo as instruções destinadas a **recuperar, atualizar, adicionar** ou **excluir dados** em um banco de dados. Nesse módulo vão aparecer comandos como INSERT, UPDATE e DELETE.

DCL é um acrônimo para data *control language* (linguagem de controle de dados). Fazem parte deste grupo as instruções destinadas a **manter a segurança adequada para o banco de dados**. Nesse módulo vão aparecer comandos como GRANT e REVOKE.

O CRUD no banco de dados SQLite

Quando o assunto é banco de dados, um termo muito comum é o CRUD, um acrônimo para as quatro operações de DML que podemos fazer em uma tabela no banco de dados – podemos inserir informações (**create**), ler (**read**), atualizar (**update**) e apagar (**delete**).



Fonte: Shutterstock.

Os passos necessários para efetuar uma das operações do CRUD são sempre os mesmos:

1. Estabelecer a conexão com um banco
2. Criar um cursor e executar o comando
3. Gravar a operação
4. Fechar o cursor e a conexão

» CREATE

A variável *conn* guarda a instância de conexão com o banco de dados. Agora é preciso criar um cursor para executar as instruções e, por fim, gravar as alterações com o método *commit()*.

```
cursor = conn.cursor()
cursor.execute("""
INSERT INTO fornecedor (nome_fornecedor, cnpj, cidade, estado, cep, data_cadastro)
VALUES ('Empresa A', '11.111.111/1111-11', 'São Paulo', 'SP', '11111-111', '2020-01-01')
""")
conn.commit()
```

» READ

Para ler os dados em uma tabela, também precisamos estabelecer uma conexão e criar um objeto cursor para executar a instrução de seleção. Ao executar a seleção, podemos usar o método *fetchall()*, para capturar todas as linhas mediante uma lista de tuplas.

```
cursor.execute("SELECT * FROM fornecedor")
resultado = cursor.fetchall()
for linha in resultado:
    print(linha)
```

» UPDATE

Ao inserir um registro no banco, pode ser necessário alterar o valor de uma coluna, o que pode ser feito por meio da instrução SQL UPDATE.

```
cursor.execute("UPDATE fornecedor SET cidade = 'Campinas' WHERE id_fornecedor = 5")
conn.commit()
```

» DELETE

Ao inserir um registro no banco, pode ser necessário removê-lo no futuro, o que pode ser feito por meio da instrução SQL DELETE.

```
cursor.execute("DELETE FROM fornecedor WHERE id_fornecedor = 2")
conn.commit()
```

Anotações Vídeo Aula

Linguagem de Programação

- # Cw4.1 – Introdução a blibliotecas Pandas

Uma Series é um como um vetor de dados (unidimensional), capaz de armazenar diferentes tipos de dados.

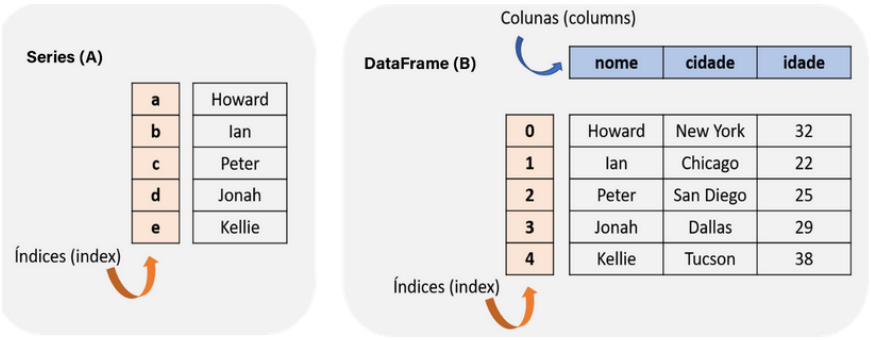
Um DataFrame é um conjunto de Series, ou, como a documentação apresenta, um contêiner para Series.

Ambas as estruturas possuem como grande característica a indexação das linhas, ou seja, cada linha possui um **rótulo** (nome) que a identifica, o qual pode ser uma string, um inteiro, um decimal ou uma data.

A figura a seguir ilustra uma **Series (A)** e um **DataFrame (B)**, veja que:

» Uma Series possui somente "uma coluna" de informação e seus rótulos (índices).

» Um DataFrame pode ter uma ou mais colunas e, além dos índices, também há um rótulo de identificação com o nome da coluna.



Series

Para construir um objeto do tipo Series, é preciso utilizar o método Series() do pacote pandas. O método possui o seguinte construtor:

```
pandas.Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)
```

Como todos os parâmetros possuem valores padrão (default), o que permite instanciar um objeto de diferentes formas. Veja algumas dessas formas:

```
pd.Series(data=5) # Cria uma Series com o valor a
pd.Series('Howard Ian Peter Jonah Kellie'.split()) # Cria uma Series com uma lista de nomes
```

DataFrame

Para construir um objeto do tipo DataFrame é preciso utilizar o método *DataFrame()* do pacote pandas. O método possui o seguinte construtor:

```
pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)
```

Veja alguns exemplos:

```
# Cria um DataFrame, de uma coluna a partir de uma lista
pd.DataFrame('Howard Ian Peter Jonah Kellie'.split(), columns=['nome'])
```

Resultado:

	nome
0	Howard
1	Ian
2	Peter
3	Jonah
4	Kellie

```
lista_nomes = 'Howard Ian Peter Jonah Kellie'.split()
lista_cpfs = '111.111.111-11 222.222.222-22 333.333.333-33 444.444.444-44
555.555.555-55'.split()
lista_emails = 'risus.varius@dictumPhasellusin.ca
Nunc@vulputate.ca fames.ac.turpis@cursusa.org non@felisullamcorper.org
eget.dictum.placerat@necluctus.co.uk'.split()
lista_idades = [32, 22, 25, 29, 38]
dados = list(zip(lista_nomes, lista_cpfs, lista_idades, lista_emails)) # cria
uma lista de tuplas # Cria um DataFrame a partir de uma lista de tuplas
pd.DataFrame(dados, columns=['nome', 'cpfs', 'idade', 'email'])
```

Resultado:

	nome	cpfs	emails	idades
0	Howard	111.111.111-11	risus.varius@dictumPhasellusin.ca	32
1	Ian	222.222.222-22	nunc@vulputate.ca	22
2	Peter	333.333.333-33	fames.ac.turpis@cursusa.org	25
3	Jonah	444.444.444-44	non@felisullamcorper.org	29
4	Kellie	555.555.555-55	eget.dictum.placerat@necluctus.co.uk	38

• Cw4.2 – Introdução a manipulação de dados Pandas

Seleção de colunas

1 . Para selecionar uma coluna usa-se a sintaxe: `meu_df['coluna']`

```
df_titanic['Age']
df_titanic['Survived']
```

2. Para selecionar mais de uma coluna é preciso passar uma lista de colunas: `meu_df[['coluna1', 'coluna2', 'coluna3']]`

```
df_titanic[['Age', 'Fare']]
df_titanic[['Name', 'PClass', 'Fare']]
```

Seleção de linhas - Filtros

Um dos recursos mais utilizados por equipes das áreas de dados é a aplicação de filtros. Imagine a seguinte situação: uma determinada pesquisa quer saber qual é a média de idade de todas as pessoas na sua sala de aula, bem como a média de idades somente das mulheres e somente dos homens. A distinção por gênero é um filtro! Esse filtro vai possibilitar comparar a idade de todos com a idade de cada grupo e entender se as mulheres ou os homens estão abaixo ou acima da média geral.

DataFrames da biblioteca pandas possuem uma propriedade chamada **loc**. Essa propriedade permite acessar um conjunto de linhas (**filtrar linhas**), por meio do índice ou por um vetor booleano (vetor de True ou False).

Saiba mais

» Ao criar uma condição booleana para os dados de uma colun, obtém-se uma Series de valores True ou False.

» Ao usar essa Series como parâmetro no loc, somente os registros que tiverem valor True são exibidos.

Exemplo:

Filtrar somente os homens que estavam a bordo e guardar dentro de um novo DataFrame.

```
filtro_homem = df_titanic['Sex'] == 'male'
df_titanic_homens = df_titanic.loc[filtro_homem]
```

Ou ainda, criar um novo DataFrame somente com os passageiros que sobreviveram:

```
filtro_sobreviventes = df_titanic['Survived'] == 1
df_titanic_sobreviventes =
df_titanic[filtro_sobreviventes]
```

O filtro sempre é criado com base em condições sobre uma ou mais colunas.

Filtros com operadores relacionais e lógicos

É possível criar filtros usando operadores relacionais e lógicos para criar condições compostas. Cada condição deve estar entre parênteses e deve ser conectada pelos operadores lógicos **AND** (**&**), **OR** (**|**).

Por exemplo, criar um novo DataFrame contendo todos os homens que sobreviveram.

```
filtro_sobreviventes = df_titanic['Survived'] == 1
filtro_homem = df_titanic['Sex'] == 'male'

df_titanic_homens_sobreviventes = df_titanic.loc[(filtro_sobreviventes) &
(filtro_homem)]
```

Ou um novo DataFrame com todos os passageiros do sexo feminino que estavam na primeira ou segunda classe. Nesse caso, é preciso utilizar um parênteses extra para garantir a ordem de execução: primeiro faz o **OU** entre pessoas que estavam na primeira e segunda classe e depois faz o **E** com pessoas do sexo feminino.

```
filtro_mulher = df_titanic['Sex'] == 'female'
filtro_classe1 = df_titanic['Pclass'] == 1
filtro_classe2 = df_titanic['Pclass'] == 2

df_titanic_mulheres_c1_c2 = df_titanic.loc[(filtro_mulher) & ((filtro_classe1) |
(filtro_classe2))]
df_titanic_mulheres_c1_c2
```

- # Cw4.3 – Visualização de dados

Figura e eixo

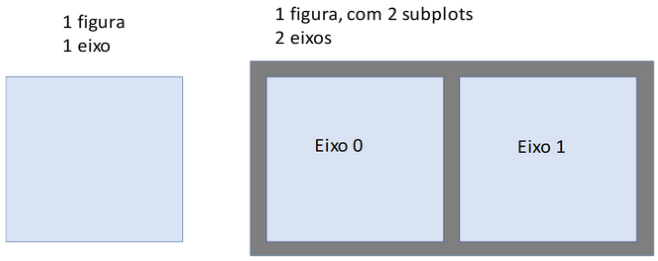
Ao utilizar a biblioteca matplotlib é preciso entender o conceito de figura e eixo (axes).

Ao criar uma figura estamos criando um espaço (tipo uma tela em branco) para se plotar o gráfico nesse espaço. Naturalmente, uma figura pode ter "subfiguras", ou seja, uma figura pode ser dividida.

» Quando uma figura possui um único espaço, ela também possui um único eixo para plotagem.

» Quando uma figura possui duas divisões, ela possui dois eixos; quando possui três divisões, três eixos, e assim por diante.

Figura e eixo matplotlib



Fonte: elaborada pela autora.

Ao se falar em criação de gráficos em Python, o profissional precisa conhecer a biblioteca matplotlib, pois diversas outras são construídas a partir desta.

Método `plt.plot(dados)`

Existem algumas opções de sintaxe para se criar figuras e eixos. Podemos deixar que a própria biblioteca gerencie para nós, dessa forma basta utilizar o método `plt.plot(dados)` que o gráfico será construído sobre uma figura e um eixo criados automaticamente.

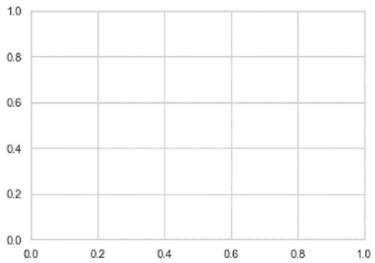
A forma automática, embora prática, não permite que o desenvolvedor tenha controle sobre o eixo que gostaria de plotar, nesse caso o ideal é criar, explicitamente, uma figura com os eixos, usando a função `plt.subplots()`.

Para criar uma figura com um eixo, de forma explícita, usamos:

```
fig, ax = plt. Subplots (1, 1)
```

Resultado:

Figura com 1 eixo criado de forma explícita

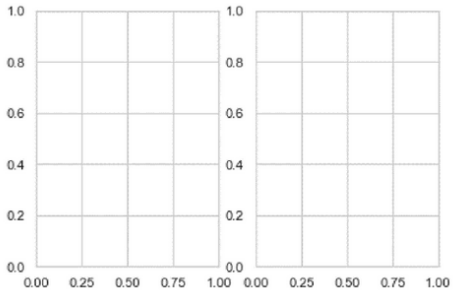


Para criar uma figura com dois eixos (1 linha, 2 colunas), de forma explícita, usamos:

```
fig, ax = plt. Subplots (1, 2)
```

Resultado:

Figura com 2 eixos criados de forma explícita



Ao criar uma figura com mais de um eixo, temos que informar em qual vamos criar um gráfico. A variável "ax", criada no último exemplo, é um vetor de eixos, ou seja, podemos acessar cada eixo pelo índice, começando por 0.

Portanto, para plotar na figura mais à esquerda escolhemos `ax[0].plot()` e na da direita, `ax[1].plot()`.

Veja a seguir:

```
fig, ax = plt. Subplots (1, 2)

Dados = range(5)

ax[0].plot(dados)
ax[1].plot(dados)
```

Plotagem sobre eixos

```
[<matplotlib.lines.Line2D at 0x2843cbfca20>]
```