

# Aula 02

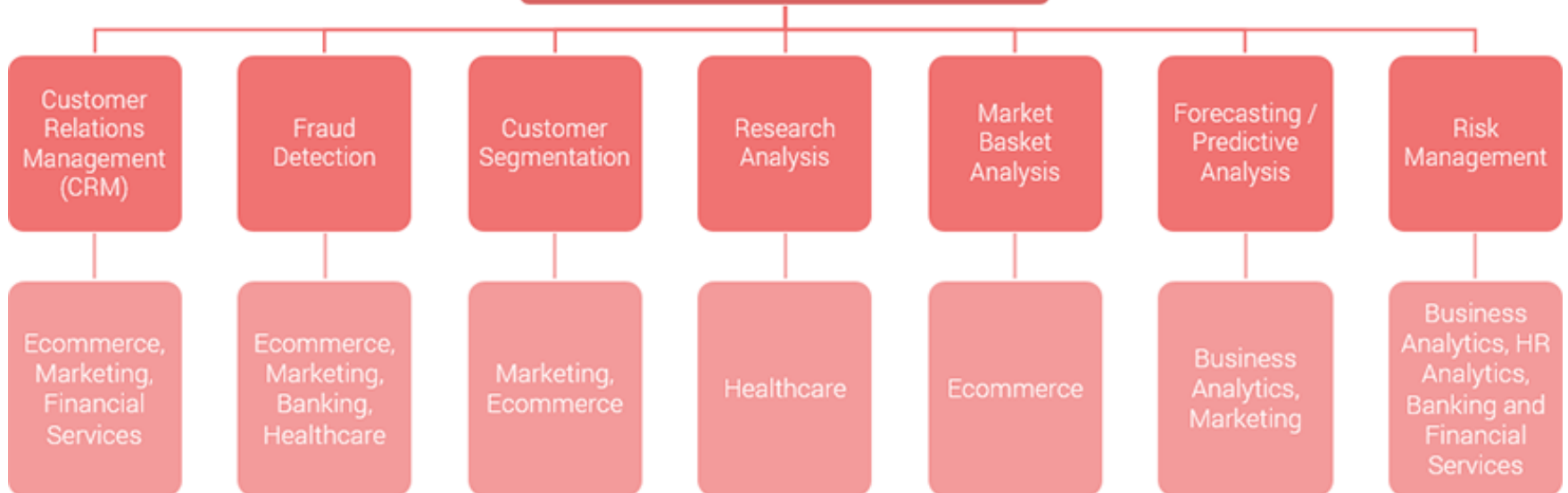
# Data Mining

Prospecção de dados ou mineração de dados é o processo de explorar dados à procura de padrões consistentes, como regras de associação ou sequências temporais, para detectar relacionamentos sistemáticos entre variáveis, detectando assim novos subconjuntos de dados.





## Data Mining Cases



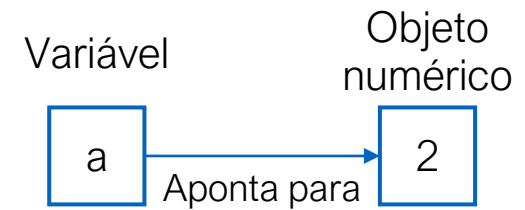


# Voltando para Python...

# Variáveis

# Declarar variáveis

```
>>> a = 2      Declarar uma variável  
>>> print(a)  
2  
>>> type(a)    Verificar tipo de variável  
<class 'int'>
```



É atribuído à variável **a** o objecto **2**, que é do tipo numérico (integer).

Uma **variável Python** é um nome simbólico, sendo uma **referência/ponteiro para um objeto**. Quando um objeto é atribuído a uma variável, pode referir-se ao objeto com esse nome.



# Declarar variáveis e tipos de variáveis

```
>>> a = 2
>>> print(a)
2
>>> type(a)
<class 'int'>
```

```
>>> a = 2
>>> b = 2.5
>>> c = "Hi there!"
>>> d = 2+3j
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'str'>
>>> type(d)
<class 'complex'>
```

Integer

Float

String

Complex

# Tipos de variáveis

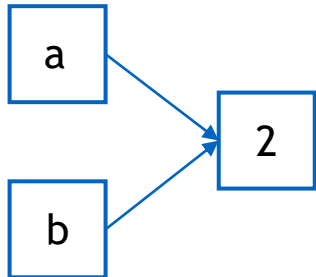
Os tipo de variáveis mais comuns e relvantes no Python são:

Tipo	Descrição	Exemplo
String	Sequência de caratères entre aspas	"Hello World!"
Int	Número inteiro	-5; 5; 10; 1904 ...
Float	Número decimal	3.2; 584.2; -35.02 ...
Complex	Numero imaginário / compelxo	3-2j; -4+2j ...
Bool	Boolean	True ou False

# Declarar variáveis

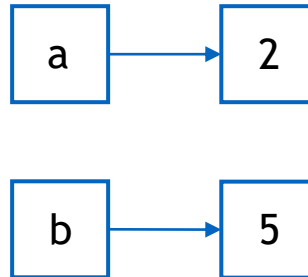
Podemos ter varias variáveis atribuidas ao mesmo valor

```
>>> a = 2
>>> b = 2
```



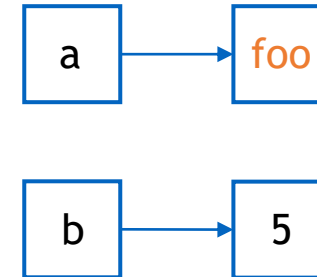
Podemos reatribuir variáveis:

```
>>> b = 5
```



Agora **a** é uma string:

```
>>> a = "foo"
```



Deixou de existir uma referência ao objeto integer 2, deixando de haver forma de o chamar.

# Typecast variáveis

Podemos mudar manualmente o tipo da variável. Por exemplo, de float para int:

```
a = 99.999
```

```
print(type(a))
```

```
<class 'float'>
```

```
b = int(a)
```

```
print(b)
```

```
print(type(b))
```

```
99
```

```
<class 'int'>
```

# Typecast variáveis

Também podemos transformar uma string numa variável numérica. Mas o número da string precisa de estar no formato correto.

```
int("99.9")
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-109-bdf6f09a0f77> in <module>()  
----> 1 int("99.9")
```

```
ValueError: invalid literal for int() with base 10: '99.9'
```

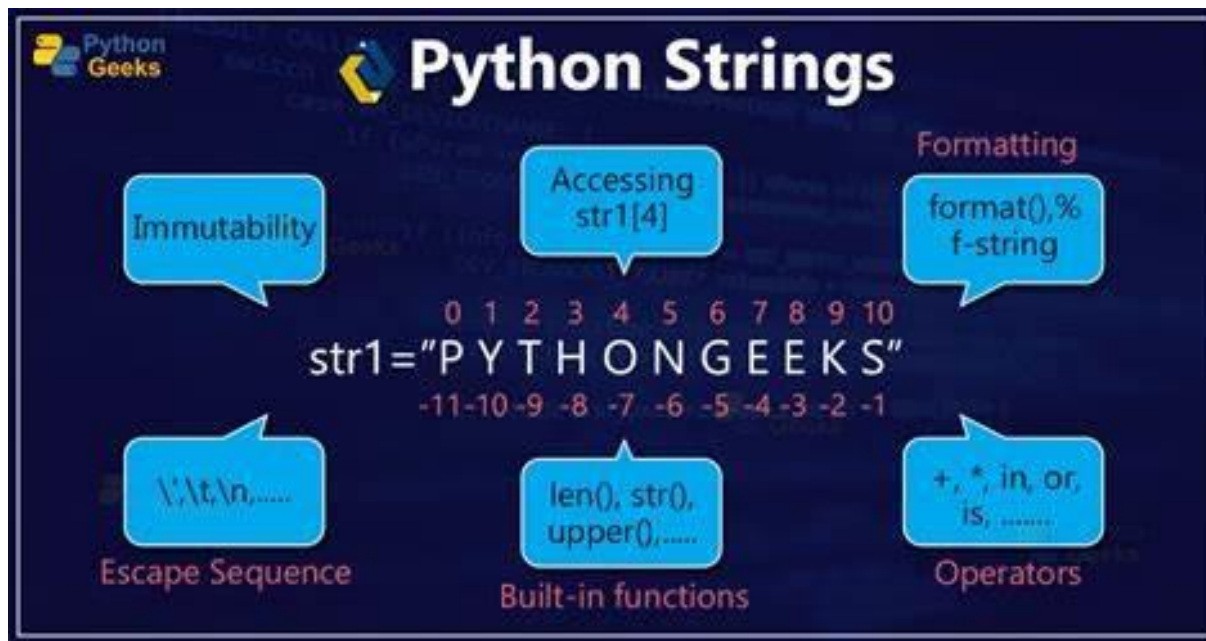
```
int("99")
```

```
99
```

```
float("99.9")
```

```
99.9
```

# String



[Strings in Python - Python Geeks](#)

# Operators

# Operators

Os operadores são símbolos especiais em Python que realizam cálculos aritméticos ou lógicos entre objectos. O valor sobre o qual o operador opera é chamado de operando.



# Arithmetic operators

Os operadores são símbolos especiais em Python que realizam cálculos aritméticos ou lógicos entre objectos. O valor sobre o qual o operador opera é chamado de operando.

Operador	Nome	Exemplo
+	Adição	$1 + 2 = 3$
-	Subtração	$3 - 2 = 1$
*	ultiplicação	$4 * 4 = 16$
/	Divisão	$16 / 4 = 4$
%	Modulus	$15 \% 4 = 3$
**	Exponciação	$3 ** 2 = 9$
//	Floor division	$10 // 3 = 3$

# Assignment operators

Os operadores são símbolos especiais em Python que realizam cálculos aritméticos ou lógicos entre objectos. O valor sobre o qual o operador opera é chamado de operando.

Operador	Exemplo	O mesmo que
+	<code>x += 3</code>	<code>x = x + 3</code>
-	<code>x -= 3</code>	<code>x = x - 3</code>
*	<code>x *= 3</code>	<code>x = x * 3</code>
/	<code>x /= 3</code>	<code>x = x / 3</code>
%	<code>x %= 3</code>	<code>x = x % 3</code>
**	<code>x **= 3</code>	<code>x = x ** 3</code>
//	<code>x //= 3</code>	<code>x = x // 3</code>

# Comparison & Logical operators

O resultado destas operações é um boolean (True ou False):

Comparison

Operador	Descrição	Exemplo
==	Igual	x == y
!=	Diferente	x != y
>	Maior que	x > y
<	Menor que	x < y
>=	Maior ou igual que	x >= y
<=	Menor ou igual que	x <= y

Logical

Operador	Descrição	Exemplo
and	True se ambas as condições forem verdadeiras	x < 5 and x < 10
or	True se pelo menos uma condição é verdadeira	x < 5 or x < 4
not	Inverte o resultado: False se o resultado for True	not(x < 5 and x < 10)

# Identity & Membership operators

O resultado destas operações é um boolean (True ou False):

Identity	Operador	Descrição	Exemplo
	is	True se ambas as variáveis forem o mesmo objeto	x is y
	is not	True se as variáveis forem objetos diferentes	x is not y
Membership	Operador	Descrição	Exemplo
	in	True se o valor especificado estiver presente no objeto pretendido	x in [x, y, z]
	not in	True se o valor especificado não estiver presente no objeto pretendido	x not in [w, y, z]

# Estruturas de dados

(listas, sets, dicionários e tupels)

# Estruturas de dados

## Lista

`a = [1, 'a', 3, 8, 6, 5]`

Uma lista é uma sequência mutável e de comprimento variável de objectos Python. Os dados são representados entre parênteses rectos.

## Dicionário

`a = {'a': 1, 'b': "xyz", 'c': 3, 'd': 4}`

É uma coleção de pares key-value mutáveis de tamanho flexível, em que key-value são objectos Python, onde cada key está associada a apenas um value. O dicionário é representado entre parênteses rectos.

## Tuple

`a = (1, 'a', 3, 8, 6, 5)`

Um tuple é uma sequência imutável e de comprimento fixo de objectos Python. Os dados são representados entre parênteses.

## Set

`a = {1, 'a', 3, 8, 6, 5}`

Um set é uma coleção de elementos únicos, não duplicáveis. São como os dicionários, mas apenas com as chaves. Os dados são representados entre chavetas.

# Estruturas de dados

## Lista

```
>>> a = [1, 'a', 3, 8, 6, 5]  
>>> type(a)  
<class 'list'>
```

## Tuple

```
>>> a = (1, 'a', 3, 8, 6, 5)  
>>> type(a)  
<class 'tuple'>
```

## Dicionário

```
>>> a = {'a':1, 'b':2, 'c':3, 'd':4}  
>>> type(a)  
<class 'dict'>
```

## Set

```
>>> a = {1, 'a', 3, 8, 6, 5}  
>>> type(a)  
<class 'set'>
```

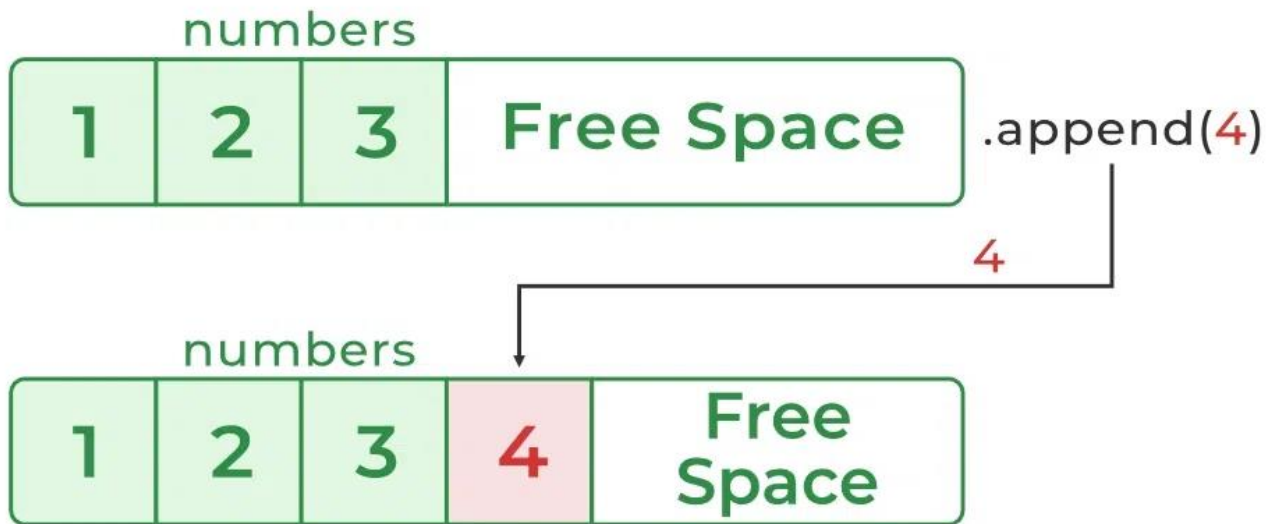
# Estruturas de dados

Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	[ ] or list()	[5.7, 4, 'yes', 5.7]
Tuple	Yes	No	( ) or tuple()	(5.7, 4, 'yes', 5.7)
Set	No	Yes	{ }* or set()	{5.7, 4, 'yes'}
Dictionary	No	Yes**	{ } or dict()	{'Jun': 75, 'Jul': 89}

[Data Structures- Lists, Tuples, Dictionaries, and Sets in Python | by Rachel Aitaru | Medium](#)



# Append - list



[Python List append\(\) Method](#)

# Dict → List

main.py

```
my_dict = {'name': 'Alice', 'age': 30}

my_list = []

dict_copy = my_dict.copy() # 👉 create copy

my_list.append(dict_copy)

print(my_list) # 👉 [{'name': 'Alice', 'age': 30}]
```

[How to Append a Dictionary to a List in Python | bobbyhadz](#)

# Dict → List

main.py

```
my_dict = {'name': 'Alice', 'age': 30}

my_list = []

my_list.append(my_dict)

my_list[0]['name'] = 'Bob'

print(my_list) # 👉 [{'name': 'Bob', 'age': 30}]

print(my_dict) # 👉 {'name': 'Bob', 'age': 30}
```

[How to Append a Dictionary to a List in Python | bobbyhadz](#)

# Listas - métodos

Method	Description
List append()	Add Single Element to The List
List extend()	Add Elements of a List to Another List
List insert()	Inserts Element to The List
List remove()	Removes Element from the List
List index()	returns smallest index of element in list
List count()	returns occurrences of element in a list
List pop()	Removes Element at Given Index
List reverse()	Reverses a List
List sort()	sorts elements of a list
List copy()	Returns Shallow Copy of a List
List clear()	Removes all Items from the List

Method	Description
list()	Creates a List

# Tuples - métodos

Method	Description
Tuple count()	returns occurrences of element in a tuple
Tuple index()	returns smallest index of element in tuple

Method	Description
tuple()	Creates a Tuple

# Sets - métodos

Method	Description
Set remove()	Removes Element from the Set
Set add()	adds element to a set
Set copy()	Returns Shallow Copy of a Set
Set clear()	remove all elements from a set
Set difference()	Returns Difference of Two Sets
Set difference_update()	Updates Calling Set With Intersection of Sets
Set discard()	Removes an Element from The Set
Set intersection()	Returns Intersection of Two or More Sets
Set intersection_update()	Updates Calling Set With Intersection of Sets
Set isdisjoint()	Checks Disjoint Sets
Set issubset()	Checks if a Set is Subset of Another Set
Set pop()	Removes an Arbitrary Element
Set symmetric_difference()	Returns Symmetric Difference
Set symmetric_difference_update()	Updates Set With Symmetric Difference
Set union()	Returns Union of Sets
Set update()	Add Elements to The Set.

Method	Description
set()	Creates a Set

# Dicionários - métodos

Method	Description
Dictionary clear()	Removes all Items
Dictionary copy()	Returns Shallow Copy of a Dictionary
Dictionary fromkeys()	creates dictionary from given sequence
Dictionary get()	Returns Value of The Key
Dictionary items()	returns view of dictionary's (key, value) pair
Dictionary keys()	Returns View Object of All Keys
Dictionary popitem()	Returns & Removes Element From Dictionary
Dictionary setdefault()	Inserts Key With a Value if Key is not Present
Dictionary pop()	removes and returns element having given key
Dictionary values()	returns view of all values in dictionary
Dictionary update()	Updates the Dictionary

Method	Description
dict()	Creates a Dictionary

# Print (Output): F-String Format

## O que são f-strings em Python?

*f-strings (formatted string literals) são uma maneira de incorporar expressões dentro de strings literais em Python, usando chaves {}. Elas fornecem uma maneira fácil e legível de formatar strings dinamicamente.*

```
nome = "Alice"  
idade = 30  
frase = f"Meu nome é {nome} e tenho {idade} anos."  
print(frase)  
Resultado:  
Meu nome é Alice e tenho 30 anos.
```

[f-strings in Python - GeeksforGeeks](https://www.geeksforgeeks.org/f-strings-in-python/)



# Print (Output): F-String Format

## O que são f-strings em Python?

*f-strings (formatted string literals) são uma maneira de incorporar expressões dentro de strings literais em Python, usando chaves {}. Elas fornecem uma maneira fácil e legível de formatar strings dinamicamente.*

```
nome = "Alice"  
idade = 30  
frase = f"Meu nome é {nome} e tenho {idade} anos."  
print(frase)  
Resultado:  
Meu nome é Alice e tenho 30 anos.
```

[f-strings in Python - GeeksforGeeks](https://www.geeksforgeeks.org/f-strings-in-python/)

# Print (Output): F-String Format

*.2f é usado para formatar números de ponto flutuante para duas casas decimais ao imprimir ou formatar strings. Por exemplo:*


```
num = 3.14159
formatted = f"{num:.2f}"
print(formatado) # Saída: 3.14
```

[f-strings in Python - GeeksforGeeks](https://www.geeksforgeeks.org/f-strings-in-python/)

# Print (Output): F-String Format

## Exemplo 1: Exibindo uma lista com f-string


python

 Copy code

```
menu = ["Pizza", "Hambúrguer", "Salada", "Sopa"]  
print(f"O menu de hoje é: {menu}")
```

Saída:


less

 Copy code

```
O menu de hoje é: ['Pizza', 'Hambúrguer', 'Salada', 'Sopa']
```

Se quiser exibir os itens formatados, você pode usar `"", ".join(lista)` (válido apenas para listas de strings):


python

 Copy code

```
print(f"O menu de hoje é: {'', '.join(menu)}")
```

Saída:

css


 Copy code

```
O menu de hoje é: Pizza, Hambúrguer, Salada, Sopa
```

# Print (Output): F-String Format

## Exemplo 2: Exibindo um dicionário com f-string


python

 Copy code

```
estoque = {"Pizza": 5, "Hambúrguer": 3, "Salada": 2}  
print(f"O estoque atual é: {estoque}")
```

Saída:


css

 Copy code

```
O estoque atual é: {'Pizza': 5, 'Hambúrguer': 3, 'Salada': 2}
```

Se você quiser formatar o dicionário como texto amigável, pode usar um loop para mostrar cada chave e valor:


python

 Copy code

```
print("Estoque atual:")  
for item, quantidade in estoque.items():  
    print(f" - {item}: {quantidade}")
```

Saída:

markdown

 Copy code

```
Estoque atual:  
- Pizza: 5  
- Hambúrguer: 3  
- Salada: 2
```

# Print (Output): F-String Format

## Exemplo 3: Exibindo uma lista de dicionários com f-string

Se você tiver uma lista de dicionários, pode iterar sobre ela:

```
python                                                                    Copy code

pedidos = [{"cliente": "João", "item": "Pizza"}, {"cliente": "Maria", "item": "Salada"}]

for pedido in pedidos:
    print(f"Cliente: {pedido['cliente']}, Item: {pedido['item']}")
```

Saída:

```
yaml                                                                    Copy code

Cliente: João, Item: Pizza
Cliente: Maria, Item: Salada
```

# Vamos praticar?

[Aula02.ipynb - Colab](#)