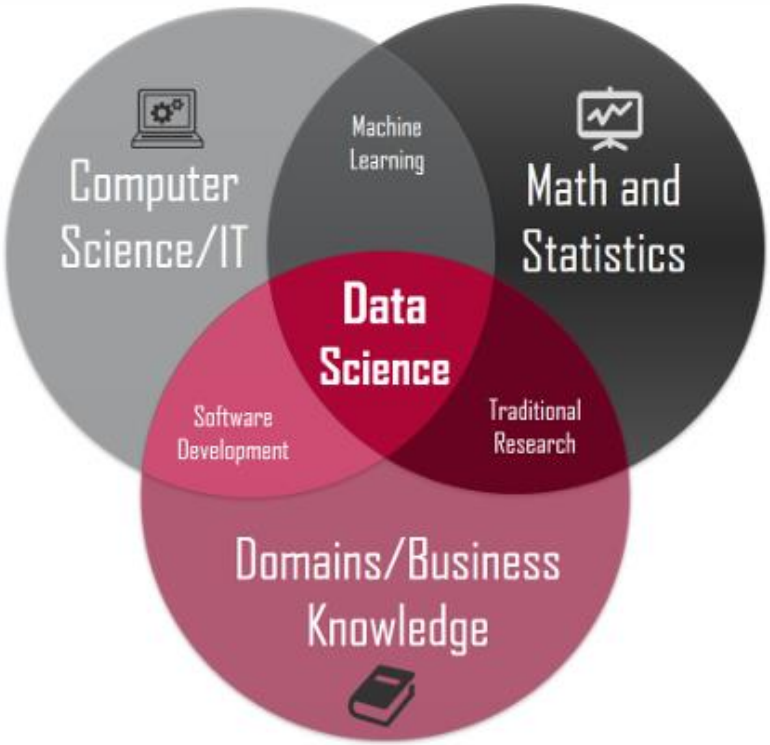
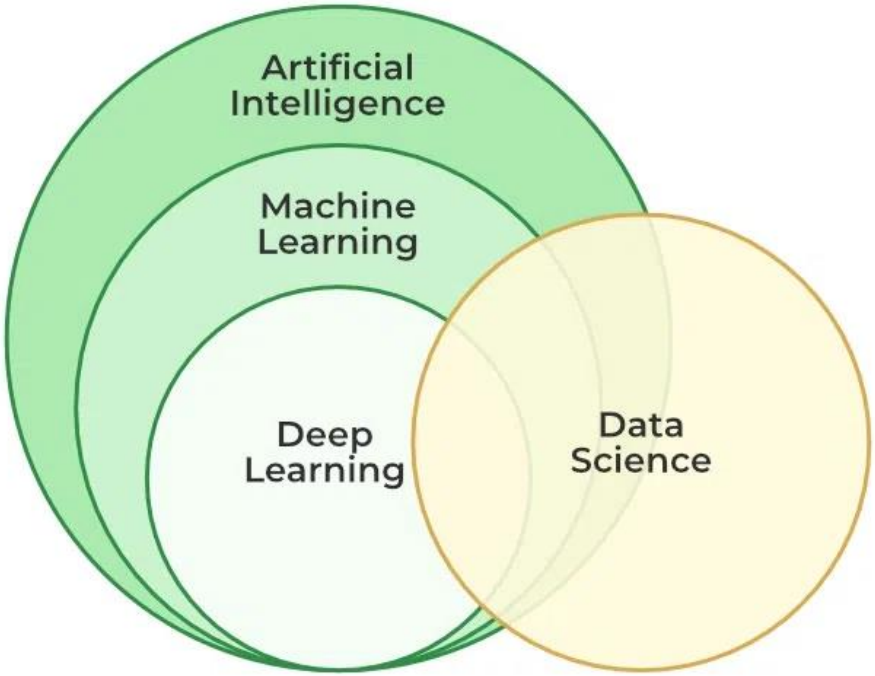
Aula 04

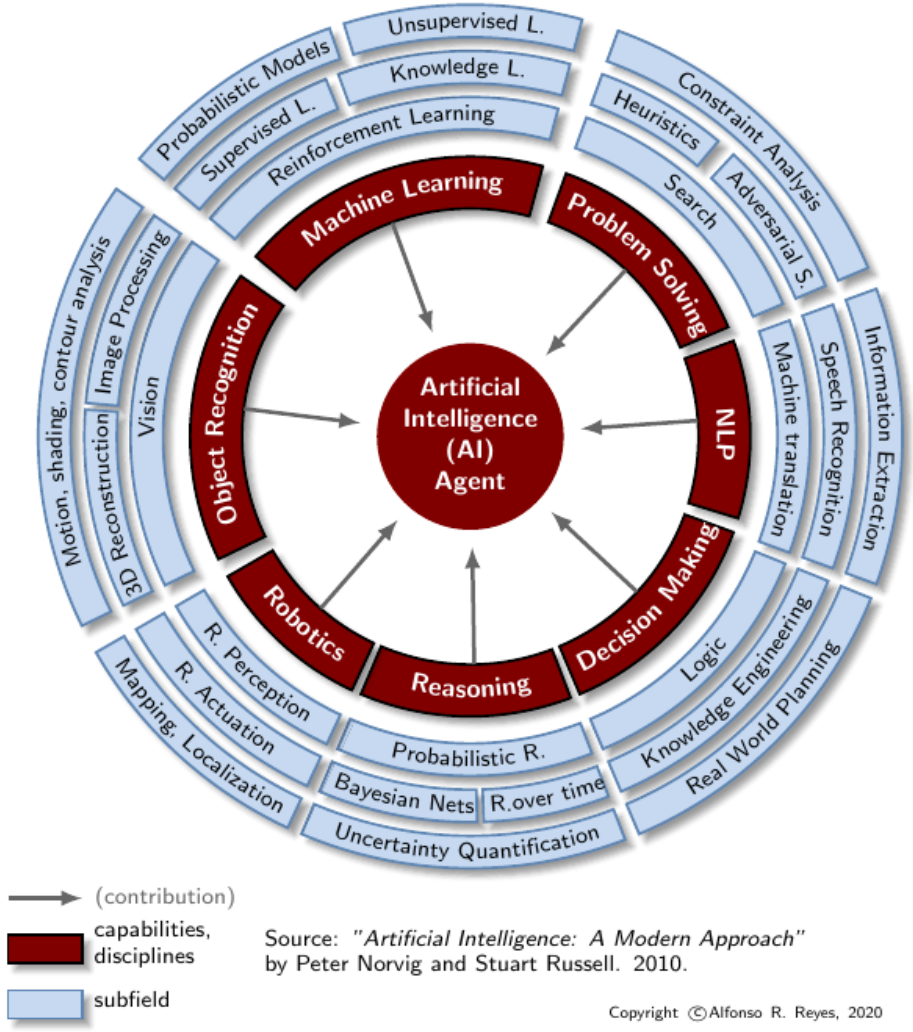


Definição: Data science é o estudo dos dados para extrair insights significativos para negócios. É uma abordagem multidisciplinar que combina práticas das áreas de matemática, estatística, inteligência artificial e engenharia da computação para analisar grandes quantidades de dados. (Amazon Web Services)

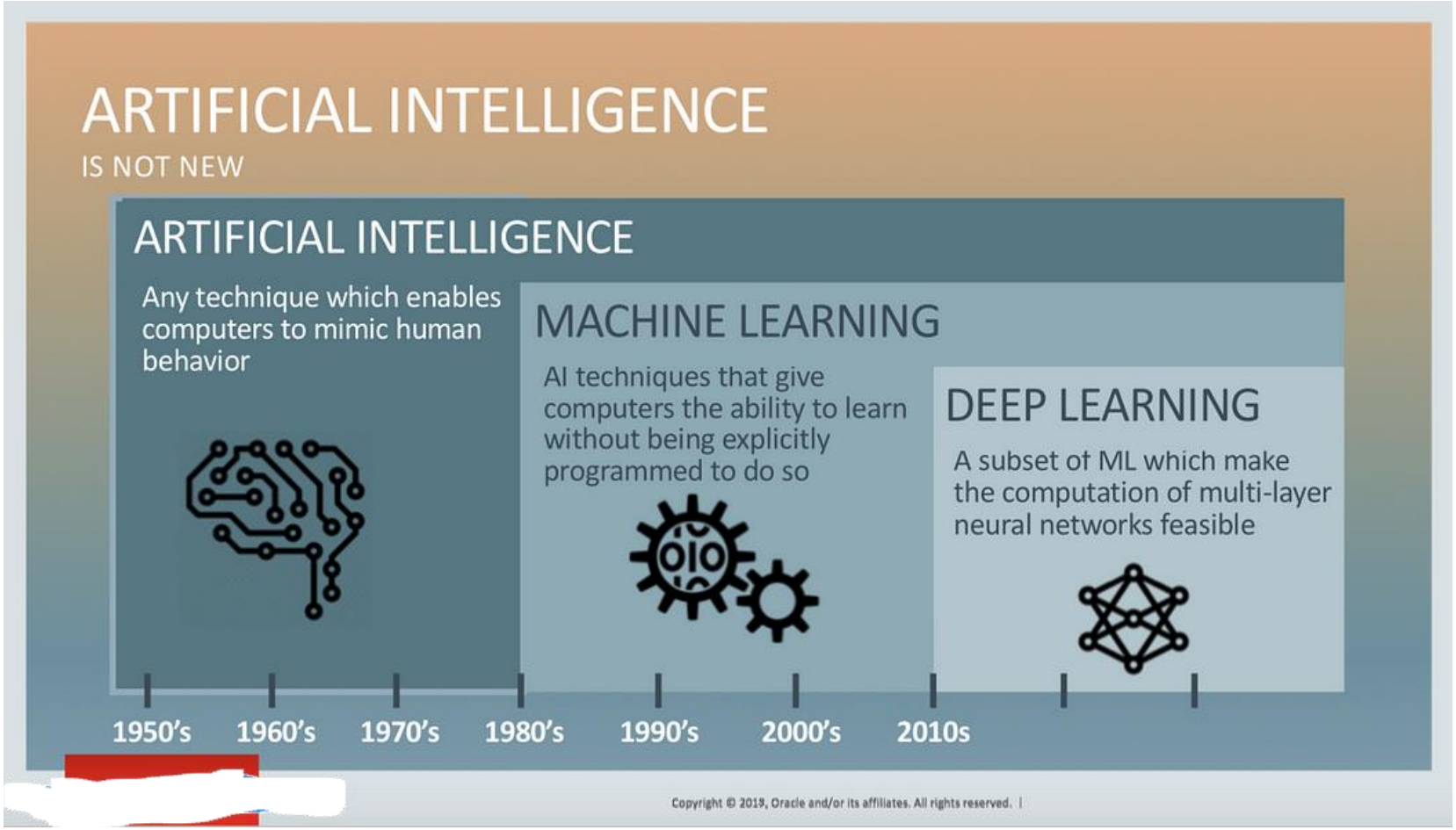
Inteligência Artificial vs Machine Learning



https://www.geeksforgeeks.org/ml-machine-learning/



https://oilgains.medium.com/why-machine-learning-is-not-artificial-intelligence-61b174a3c9a2



https://oilgains.medium.com/why-machine-learning-is-not-artificial-intelligence-61b174a3c9a2


TOP PYTHON MACHINE LEARNING LIBRARIES
pandas
pandas
SciPy
NumPy
Matplotlib
K
Keras
Tensor Flow
PyTorch
learn
Scikit-learn
theano
theano
orange
Orange3

Voltando para Python…

Funções

Funções

O Python inclui várias funções embutidas, como por exemplo **print()**. No entanto, pode ser útil **alargar as funcionalidades do nosso código** definindo as nossas próprias funções.

Funções

O Python inclui várias funções embutidas, como por exemplo **print()**. No entanto, pode ser útil **alargar as funcionalidades do nosso código** definindo as nossas próprias funções.

O que são funções? São **blocos de código que executam operações quando chamadas.**

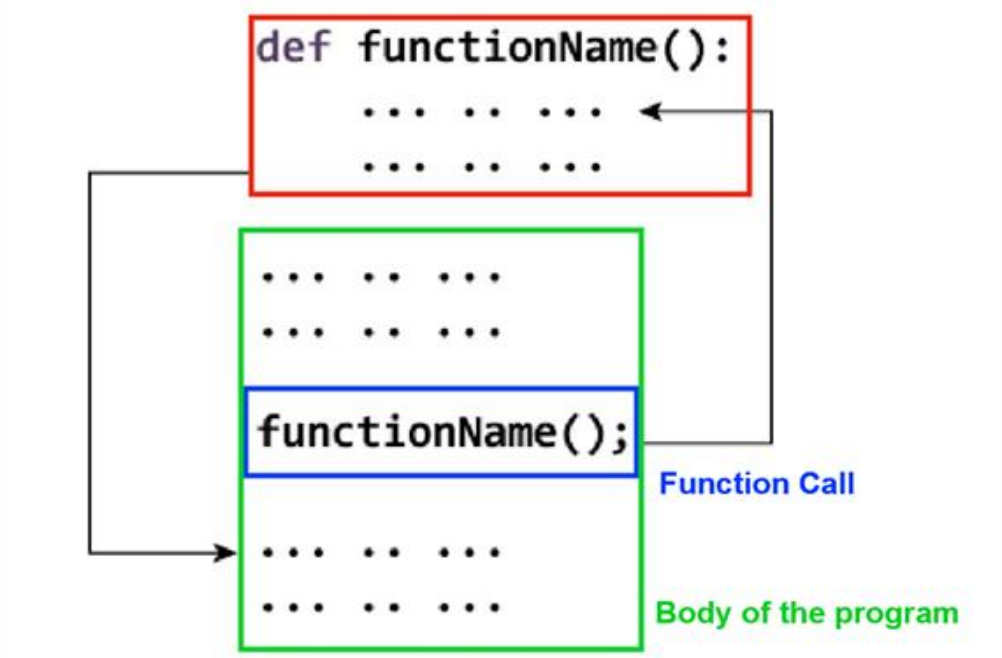
Funções

O Python inclui várias funções embutidas, como por exemplo **print()**. No entanto, pode ser útil **alargar as funcionalidades do nosso código** definindo as nossas próprias funções.

O que são funções? São **blocos de código que executam operações quando chamadas.**

As funções permitem-nos **dividir uma tarefa grande**, com elementos repetitivos, **em blocos mais pequenos**. Além disso, escrever funções **facilita o processo de debug** do nosso código.

Funções – como funcionam?



Funções – exemplos

Declaração da Função

```
def hello():
    print("hello world")
```

```
hello
```

```
<function __main__.hello()>
```

Chamar a Função

```
hello()
```

```
hello world
```

Funções – exemplos

```
def multiply(a,b):
    output = a*b
    return output

product = multiply(2,2)
print(product)
```

Declaração da Função

Chamar função e guardar output numa variável

```
4
```

```
def divide(a,b):
    output = a/b

division = divide(2,4)
print(division)
```

Declaração da Função

Chamar função e guardar output numa variável. Mas aqui a variável assuma o valor None. Isto porque dentro da função não existe um return.

```
None
```

Funções – exemplos

O que faz a seguinte função?

```
def function(lista, target):
    for i, value in enumerate(lista):
        if value == target:
            return i
    return -1
```

Funções – exemplos

O que faz a seguinte função?

```
def function(lista, target):
    for i, value in enumerate(lista):
        if value == target:
            return i
    return -1
```

E porque não devemos definir o if statement desta forma?

```
def function(lista, target):
    for i, value in enumerate(lista):
        if value == target:
            return i
        else:
            return -1
```

Funções – *args e **kwargs

O Python tem uma sintaxe especial, * (asterisco) e ** (asterisco duplo), que permite **passar um número variável de argumentos numa função**:

```
def function_name(arguments, *args, **kawrgs):
    #perform tasks
    return something
```

Por convenção, estes **são escritos como *args e **kwargs**, mas apenas os asteriscos são importantes. Pode-se definir igualmente *vars e **vars.

Funções – *args e **kwargs

O parâmetro ***args é utilizado para passar uma lista de argumentos** de comprimento variável **sem palavra-chave** para a sua função.

```
def greet(*args):
    for name in args:
        print("Hello", name)

greet("Monica", "Luke", "Steve", "John")
```

```
Hello Monica
Hello Luke
Hello Steve
Hello John
```

Funções – *args e **kwargs

O parâmetro ***args é utilizado para passar uma lista de argumentos** de comprimento variável **sem palavra-chave** para a sua função.

```
def multiply(*args):
    output = 1
    for n in args:
        output *= n
    return output

print(multiply(2,4,5,6))
print(multiply(12,34,54,346))
```

```
240
7623072
```

Funções – *args e **kwargs

O parâmetro **kwargs permite passar uma lista de argumentos de comprimento variável com palavra-chave na nossa função.

```
def something(**kwargs):
    for key,value in kwargs.items():
        print("The value of", key, "is", value)

something(January=1, February=2, March=3)
```

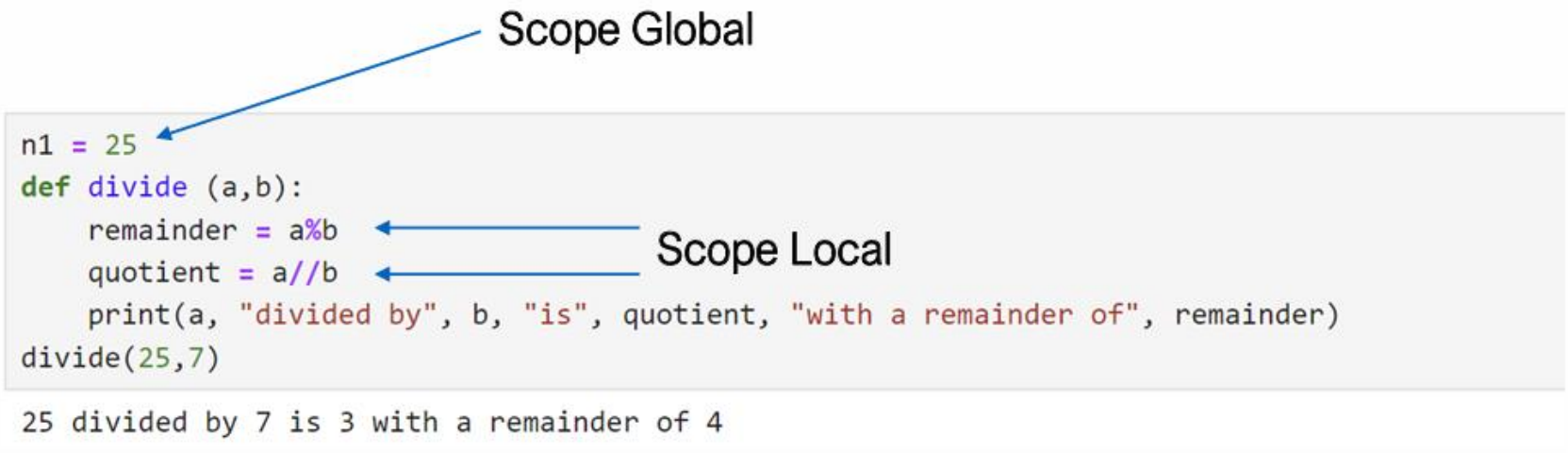
```
The value of January is 1
The value of February is 2
The value of March is 3
```

Scope de Variáveis

As variáveis podem ter um scope global ou local:

- **Variáveis Globais:** podem ser chamadas ao longo de todo o programa e dentro de todas as funções.
- **Variáveis Locais:** podem ser chamadas apenas dentro da função onde foram decalaradas.

Scope de Variáveis




```
n1 = 25
def divide (a,b):
    remainder = a%b
    quotient = a//b
    print(a, "divided by", b, "is", quotient, "with a remainder of", remainder)
divide(25,7)
```

```
25 divided by 7 is 3 with a remainder of 4
```

Scope de Variáveis

Scope Global

```
animal = "cat"
def change_and_print_global():
    animal = "dog"
    print("after the change:", animal)
change_and_print_global()
```

```
after the change: dog
```

Scope Local

```
animal = "cat"
def change_and_print_global():
    print("inside change_and_print_global:", animal)
    animal = "dog"
    print("after the change:", animal)
change_and_print_global()
```

Deu erro pois tentámos imprimir a variável animal antes de esta ser definida localmente

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In[55], line 6
      4     animal = "dog"
      5     print("after the change:", animal)
----> 6 change_and_print_global()

Cell In[55], line 3, in change_and_print_global()
      2 def change_and_print_global():
----> 3     print("inside change_and_print_global:", animal)
      4     animal = "dog"
      5     print("after the change:", animal)

UnboundLocalError: cannot access local variable 'animal' where it is not associated with a value
```

Scope de Variáveis

Scope Global

Scope Local

```
animal = 'cat'
def change_and_print_global():
    animal = 'dog'
    print('after the change:', animal)
change_and_print_global()
print(animal)
```

```
after the change: dog
cat
```

Ao chamar a função, a variável animal definida dentro da função (outrora local), passa a ser global, podendo ser altera da mesmo dentro da função

```
animal = "cat"
def change_and_print_global():
    global animal
    animal = "dog"
    print("inside change_and_print_global:", animal)
print(animal)
change_and_print_global()
print(animal)
```

```
cat
inside change_and_print_global: dog
dog
```

Funções Lambda

Por vezes, queremos **escrever uma função numa só linha** para uma tarefa específica. É aqui que entram as **funções lambda**, que têm a seguinte forma:

```
lambda arguments: expression
```

Que é o mesmo do que fazer:

```
def function_name(arguments):
    #perform tasks
    return expression
```

No entanto, as funções lambda não têm um nome.

Funções Lambda

```
add = lambda x,y: x + y
```

```
add(2,3)
```

```
lista = [1,2,3,4]
lista_sqrd = list(map(lambda x: x*2, lista))
lista_sqrd
```

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

Funções Lambda

```
add = lambda x,y: x + y
```

```
add(2,3)
```

```
lista = [1,2,3,4]
lista_sqrd = list(map(lambda x: x*2, lista))
lista_sqrd
```

1º argumento

2º argumento

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

O Map aplica uma função (1º argumento), a cada objeto de uma coleção (2º argumento). Em muitos casos, queremos aplicar uma função simples e podemos defini-la numa linha com a função lambda.

Funções Lambda

Além do map(), o filter() é outra função útil que requer a passagem de uma função lambda.

```
from random import sample
```

```
X = sample(range(-25, 25), 25)
```

```
print(X)
```

```
[-22, 18, -15, -20, 15, 14, 24, 16, 13, 10, 1, -23, -7, 19, 22, -13, 21, -11, 23, 20, -25, -4, -21, 12, -3]
```

```
X_filtered = list(filter(lambda x: x>0, X))
```

```
print(X_filtered)
```

```
[18, 15, 14, 24, 16, 13, 10, 1, 19, 22, 21, 23, 20, 12]
```

Funções Lambda

O que estão estas linhas de código a fazer? Qual o output?

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

```
list(filter(lambda x: x%2 == 0, example))
```

Bibliotecas

Como importer biblioteca?

```
import plotly.io as pio
import plotly.express as px

pio.renderers.default = "notebook"
colors = {
          'setosa': '#0099ff', # Azul
          'virginica': '#ff1a1a', # Vermelho
          'versicolor': '#5cd65c' # Verde
          }

fig = px.scatter(df, x='sepal length (cm)', y='petal length (cm)',
                 color=df['class'].map(colors), title='Dataset Iris')

fig.show()
```

25] ✓ 0.2s Python

Vamos praticar!

1. Escreva um programa Python para quadrar e cubar cada número em uma determinada lista de
inteiros usando uma função Lambda (ex: [1, 2, 3, 5, 7, 8, 9, 10]).

2. Escreva um programa Python para contar os números pares e ímpares em uma determinada lista de
inteiros usando uma função Lambda (ex: [1, 2, 3, 5, 7, 8, 9, 10]).

3. Escreva uma função que encontrará todos os números entre X e Y (incluídos) tais
que cada dígito desses números é um número par. Os números obtidos deve ser impresso numa sequência separada por vírgulas numa única linha.
Exemplo:
X=10 e Y=50.
Então a saída deve ser 20, 22, 24, 26, 28, 40, 42, 44, 46, 48.