UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE QUÍMICA

Apostila de Acompanhamento

Playlist 24: Curso de Python Básico

Canal no Youtube: Dr.Prof. Edenir Pereira Filho

André Luiz Machado Simão de Souza

São Carlos
2022

Contato

Em caso de dúvidas sobre qualquer parte deste treinamento, entre em contato através destas plataformas:

linkedin.com/in/almssimao/

alms.simao@gmail.com

Materiais complementares

O link abaixo contém um repositório que criei no GitHub, baixe o conteúdo completo do curso, isto é, inclui todos os jupyter notebooks de todas as seções, também há resumos de alguns módulos utilizados nas resoluções de exercícios e duas planilhas do Excel com dados de problemas em aplicações químicas.

O repositório existe uma pequena tabela contendo os conteúdos de cada vídeo que poderão auxiliá-lo no acompanhamento.

https://github.com/AndreSimao-alms/Curso-de-python

Último comentário

Prezados(as) leitores(as),

Depois de baixar o anaconda ou tiver logado em sua conta Google no Colaboratory, lembre-se de uma coisa: pratique os exercícios propostos nos vídeos! A fixação é importante para aprender a construir códigos, não tenha pressa e boa aula!

Atenciosamente,

André Simão.

Agradecimentos

Agradeço ao Dr. Prof. Edenir Pereira Filho pelo apoio ao projeto e pelo espaço oferecido em seu canal do YouTube.

Seção 1 - Variáveis, Operadores Matemáticos e Funções input() e print()

March 7, 2022

1 Tipo de variáveis

Vamos dar o **primeiro passo** em nossa playlist apresentando os principais **tipos de dados** que vamos utilizar aqui em nosso treinamento. Variáveis possuem algumas **diretrizes** para manter **boas práticas de codificação**, estas são: - Podem conter letras, números e *undercores* - Devem começar somente com letra ou *undercore* - Devem está em caixa-baixa (caracteres minúsculos)

2 Parte 1 - int & float

2.1 Int

- Número inteiros.

[1]:
```
# Apresentado no terminal Prompt de Comando
a = 1
b = 2
a + b
```

[1]:
```
3
```

2.2 Float

- Números decimais:
  CUIDADO! Utilize **ponto** para a separação de casas, uma vez que a vírgula é utilizada para declaração de tuplas, por exemplo:

[6]:
```
# Apresentado no terminal Prompt de Comando
a = 1
b = 3.88
b - a
```

[6]:
```
2.88
```

2.2.1 Exemplo 1 - Conversão de unidades milhas/kilometros

Aplique ao prompt de comandos a equação abaixo para a conversão de unidade de medida kilometros e milhas

```
milhas = 1.60934*distancia
```

[3]:
```
# Apresentado no terminal Prompt de Comando
distancia = 3.6 # Em kilometros
milhas = 1.60934*distancia
milhas
```

[3]: 5.793624

2.3 Função type()

Esta função recebe uma variável retorna a sua respectiva classificação.

[4]:
```
# Apresentado no terminal Prompt de Comando
type(milhas)
```

[4]: float

[7]:
```
type(a)
```

[7]: int

3 Parte 2 - Uso de Strings

3.1 String

Combinação de múltiplos caracteres, em Python, os caracteres especiais são incluídos. Basicamente, a declaração de uma string pode ser através de aspas simples, aspas duplas ou utilizando a função `str()`. Podemos também salvar em variáveis textos em **docstrings** que são definidos por aspas triplas `'''Insere o texto aqui'''` ou de aspas duplas triplas `"""Insere o texto aqui"""`.

[27]:
```
nome = 'André Luiz'
type(nome)
```

[27]: str

3.1.1 Index

O conceito de index é utilizado em diferentes **iteráveis**, este recurso permite **acessar valores** através de um **índice** que se inicia apartir de zero e que também aceita valores negativos.

[28]:
```
#[0,1,2,3,4,5,6,7,8,9,10]

nome[-2]
```

[28]: 'i'

3.1.2 Slice

Slice é um recurso utilizado em elementos iteráveis onde **seleciona intervalos** através do uso do **index**.

[29]:
```
nome[6:10]
```

[29]:
```
'Luiz'
```

3.2 Alguns métodos strings

Em Python, variáveis são **objetos** criados por **classes internas** da linguagem que **definem** o tipo de variável, estes objetos possuem **método**s, que nada mais são que **funções da sua própria classe**. Em suma, estes métodos são ferramentas essenciais para resolução de problemas. Assim, a variável string, possui uma gama de métodos para a sua manipulação, vejamos alguns exemplos abaixo:

3.2.1 upper( )

Retorna string com caracteres maiúsculos

[30]:
```
nome.upper()
```

[30]:
```
'ANDRÉ LUIZ'
```

3.2.2 lower( )

Retorna string com caracteres minúsculos

[31]:
```
nome.lower()
```

[31]:
```
'andré luiz'
```

3.2.3 title( )

Retorna string com primeiro caractere maiúsculo de cada palavra

[32]:
```
'curso de python'.title()
```

[32]:
```
'Curso De Python'
```

3.2.4 split( ) e parâmetro “sep”

Retorna uma lista compostas pelas palavras construídas em um string. Outro novo elemento apresentado aqui é o conceito de **parâmetro**, estes são entradas (*inputs*) que podem ser obrigatórios ou não para um determinado método ou função.

[33]:
```
curso = 'curso_de_python'
curso.split(sep='o')
```

[33]:
```
['curs', '_de_pyth', 'n']
```

3.2.5 replace( )

Recebe (elemento_velho, elemento_novo) e retorna uma nova string

[34]:
```
curso = 'curso_de_python'
curso = curso.replace('_',' ')
curso.title()
```

[34]:
```
'Curso De Python'
```

3.2.6 map( )

Aplica uma função para cada iterável. Neste exemplo vamos converter uma docstring em uma lista contendo strings, então, através da função map(), poderemos converter para variável float com um comando em todos itens desta lista.

[35]:
```
dados = '''19.37
33.87
12.17
22.86
10.14
15.41
6.31
7.77
46.61
69.45
31.21
32.82
34.74
40.02
26.23
31.21'''
dados = dados.replace('\n', ',')
dados = dados.split(sep= ',')
dados = list(map(float,dados))
print(dados)
```

```
[19.37, 33.87, 12.17, 22.86, 10.14, 15.41, 6.31, 7.77, 46.61, 69.45, 31.21,
32.82, 34.74, 40.02, 26.23, 31.21]
```

4 Desafio 1 - Contador de caracteres e palavras

Construa um programa que receba uma string e calcule: - O número de palavras - O número de caracteres - O número de caracteres sem espaços

Utilize a função `len()`

[36]:
```
dados_n = 'Return True if the object argument is an instance of the classinfo␣
 ↪argument, or of a (direct, indirect, or virtual) subclass thereof. If object␣
 ↪is not an object of the given type, the function always returns False. If␣
 ↪classinfo is a tuple of type objects (or recursively, other such tuples) or␣
 ↪a Union Type of multiple types, return True if object is an instance of any␣
 ↪of the types. If classinfo is not a type or tuple of types and such tuples,␣
 ↪a TypeError exception is raised.'
n_palavras = len(dados_n.rsplit())
n_carac = len(dados_n)
n_carac_sem_espaco = len(dados_n.replace(' ', ''))
print(n_palavras)
print(n_carac)
print(n_carac_sem_espaco)
```

```
87
478
392
```

4.0.1 Exemplo 2 (Extra) | Limpeza de dados

4.0.2 Efeitos de Planejamento de Experimentos 2^4

Este exemplo não foi apresentado nas gravações da playlist devido o conteúdo necessário para a construção do código. No entanto, está exposto aqui nesta apostila para demonstrar como podemos utilzar os métodos strings para a resolução de problemas pertinentes.

[54]:
```
# Tabela de efeitos de planejamento experimental 2^4
# Dados copiados diretamente de uma planilha .xlsx para um arquivo .txt
dados =␣
 ↪'''-55        -55        -55        -55        55        55        55        55        55
53        -53        -53        -53        -53        -53        -53        53        53
-91        91        -91        -91        -91        91        91        -91        -91
94        94        -94        -94        94        -94        -94        -94        -94
-88        -88        88        -88        88        -88        88        -88        88
97        -97        97        -97        -97        97        -97        -97        97
-100        100        100        -100        -100        -100        100        100        -1
98        98        98        -98        98        98        -98        98        -98        -
-72        -72        -72        72        72        72        -72        72        -72
62        -62        -62        62        -62        -62        62        62        -62
-90        90        -90        90        -90        90        -90        -90        90
90        90        -90        90        90        -90        90        -90        90        -
-90        -90        90        90        90        -90        -90        -90        -90
98        -98        98        98        -98        98        98        -98        -98
-104        104        104        104        -104        -104        -104        104        10
100        100        100        100        100        100        100        100        100
```

[55]:
```
import numpy as np
import pandas as pd # seção 5
```

```python
def limpar_dados(dados): # Função parametrizada - > seção 4
    dados = dados.replace('\t', ' ')
    dados = dados.replace('\n', ',')
    dados = dados.split(sep=',')
    dados = [[i] for i in dados]  # List comprehensions
    return np.array([dados[j][0].split() for j in range(len(dados))]) # List␣
 ↪comprehensions e função range()-> seção 3 e 4
def gerar_df(dados):
    columns = [1,2,3,4,12,13,14,23,24,34,123,124,134,234,1234]
    return pd.DataFrame(limpar_dados(dados),columns=columns) #seção 5
gerar_df(dados)
```

[55]:

	1	2	3	4	12	13	14	23	24	34	123	124	134	\
0	-55	-55	-55	-55	55	55	55	55	55	55	-55	-55	-55	
1	53	-53	-53	-53	-53	-53	-53	53	53	53	53	53	53	
2	-91	91	-91	-91	-91	91	91	-91	-91	91	91	91	-91	
3	94	94	-94	-94	94	-94	-94	-94	-94	94	-94	-94	94	
4	-88	-88	88	-88	88	-88	88	-88	88	-88	88	-88	88	
5	97	-97	97	-97	-97	97	-97	-97	97	-97	-97	97	-97	
6	-100	100	100	-100	-100	-100	100	100	-100	-100	-100	100	100	
7	98	98	98	-98	98	98	-98	98	-98	-98	98	-98	-98	
8	-72	-72	-72	72	72	72	-72	72	-72	-72	-72	72	72	
9	62	-62	-62	62	-62	-62	62	62	-62	-62	62	-62	-62	
10	-90	90	-90	90	-90	90	-90	-90	90	-90	90	-90	90	
11	90	90	-90	90	90	-90	90	-90	90	-90	-90	90	-90	
12	-90	-90	90	90	90	-90	-90	-90	-90	90	90	90	-90	
13	98	-98	98	98	-98	98	98	-98	-98	98	-98	-98	98	
14	-104	104	104	104	-104	-104	-104	104	104	104	-104	-104	-104	
15	100	100	100	100	100	100	100	100	100	100	100	100	100	

	234	1234
0	-55	55
1	-53	-53
2	91	-91
3	94	94
4	88	-88
5	97	97
6	-100	100
7	-98	-98
8	72	-72
9	62	62
10	-90	90
11	-90	-90
12	-90	90
13	-98	-98
14	104	-104
15	100	100

5 Parte 3 - Variáveis Compostas

5.1 Uma abordagem breve sobre variáveis compostas

Variáveis compostas são coleções de itens, ou seja, são variáveis que podem armazenar vários tipos de variáveis em um único elemento. As coleções que apresentarei neste módulos será as listas e dicionários. As **listas** são coleções **mutáveis**, em outras palavras podem ser modificadas, adicionando, removendo e ordenando os seus itens.

5.2 list

[56]:
```
lista = [True, 2, 2.5, 'andre', [1,2,3], (2,3)]
lista
tuplas = True, 2, 2.5, 'andre', [1,2,3], (2,3)
print(type(tuplas))
```

```
<class 'tuple'>
```

5.2.1 index

```
index ->[0,1,2,3,4]

lista = [1,2,3,4,5]

index->[-5,-4,-3,-2,-1]

lista[0]
>>> 1

lista[-1]
>>> 5
```

[57]:
```
lista[3]
lista[-1][1]
```

[57]: 3

5.2.2 slice

```
index ->[0,1,2,3,4]
lista = [1,2,3,4,5]

lista[0:3]
>>> 1
```

[58]:
```
lista = [1,2,3,4,5]
lista[0:4]
```

[58]: [1, 2, 3, 4]

5.2.3 Método append ()

Retorna um valor para o final de uma lista

[59]:
```
lista.append([1,2,3])
lista
```

[59]:
```
[1, 2, 3, 4, 5, [1, 2, 3]]
```

5.3 dict

5.3.1 Estrutura de dicionários:

```
#ESTRUTURA GENÉRICA:

dict_ex = {'Key':value}

dicionario = {'Nome': 'André', 'Idade': 25, 'Semestre': 9, 'Ano de ingresso':2017}

# items:
>>> 'Nome': 'André', 'Idade': 25, 'Semestre': 9, 'Ano de ingresso':2017

# key:
>>> 'Nome', 'Idade', 'Semestre', 'Ano de ingresso'

# values:
>>> 'André', 25, 9, 2017
```

[60]:
```
dicionario = {'Nome': 'André', 'Idade': 25, 'Semestre': 9, 'Ano de ingresso':
↪2017}
len(dicionario)
```

[60]:
```
4
```

5.3.2 Método items( )

Retorna as itens do dicionário.

[61]:
```
dicionario.items()
```

[61]:
```
dict_items([('Nome', 'André'), ('Idade', 25), ('Semestre', 9), ('Ano de
ingresso', 2017)])
```

5.3.3 Método keys( )

Retorna as chaves do dicionário.

[62]:
```
dicionario.keys()
```

[62]:
```
dict_keys(['Nome', 'Idade', 'Semestre', 'Ano de ingresso'])
```

5.3.4 Método values( )

Retorna os valores do dicionário.

[63]:
```
dicionario.values()
```

[63]:
```
dict_values(['André', 25, 9, 2017])
```

5.3.5 Método get( )

Recebe o nome da chave e retorna o seu repectivo valor. Caso o parâmetro inserido for inesxistente será retornado um NoneType

[64]:
```
type(dicionario.get('Curso'))
```

[64]:
```
NoneType
```

5.3.6 Adicionando item ao dicionário

[65]:
```
dicionario['Curso']='Química'
```

[66]:
```
dicionario['CPF']='111.222.333-44'
```

[67]:
```
dicionario
```

[67]:
```
{'Nome': 'André',
 'Idade': 25,
 'Semestre': 9,
 'Ano de ingresso': 2017,
 'Curso': 'Química',
 'CPF': '111.222.333-44'}
```

6 Parte 4 - Função print()

6.1 Função print()

- Imprime valores ou elementos no terminal do compilador

6.1.1 Exemplo 1 - Conversão de unidades milhas/kilometros

Crie um programa que receba a distância em milhas e tempo (h) e imprima valores de distância em km, tempo em horas e a valocidade escalar média em km/h.

`milhas = 1.60934*distancia`

[70]:
```
distancia = float(input())
tempo = float(input())
km = 1.60934*distancia
v_media = km/tempo
print(distancia)
```

```
print(tempo)
print(v_media)
```

```
12
2
12.0
2.0
9.65604
```

6.1.2 Concatenação de variaveis na função print

[71]:
```
print('Distância:', distancia )
print('Tempo:',tempo)
print('Velocidade média', v_media)
```

```
Distância: 12.0
Tempo: 2.0
Velocidade média 9.65604
```

6.1.3 Método 'f'

[72]:
```
print(f"Distância: {distancia}")
print(f"Tempo:' {tempo}")
print(f"Velocidade média' {v_media}")
```

```
Distância: 12.0
Tempo:' 2.0
Velocidade média' 9.65604
```

6.1.4 Método .format()

[73]:
```
print("Distância: {}, {}".format(distancia,tempo))
print("Tempo:' {}".format(tempo))
print("Velocidade média' {}".format(v_media))
```

```
Distância: 12.0, 2.0
Tempo:' 2.0
Velocidade média' 9.65604
```

6.1.5 %s | %f | %d

[74]:
```
print("Distância: %d"%distancia)
print("Tempo:' %d"%tempo)
print("Velocidade média' %.1f"%v_media)
print('A distãncia percorrida foi de %d km e sua respectiva velocidade média␣
 ↪foi de %.2f km/h'%(distancia, v_media))
```

```
Distância: 12
Tempo:' 2
```

```
Velocidade média' 9.7
A distância percorrida foi de 12 km e sua respectiva velocidade média foi de
9.66 km/h
```

7 Parte 5 - Função Input()

7.1 Função input()

- insere dados ao programa
- Estrutura: _tipodevariavel_input_informação

Função input( ) documentação:
https://docs.python.org/3/library/functions.html#input

[76]:
```
a = int(input('Digite um valor: '))
```

```
Digite um valor: 12
```

[77]:
```
type(a)
```

[77]:
```
int
```

7.1.1 Exemplo 1 - Construção de tanque cilídrico inox

Construa um programa que leia diâmetro e altura de um cilíndro e retorna o usuário valor e metragem das chapas inox necessárias para construção do cilindro. - Preço por m² da chapa: R$580,00 - A empresa realiza cortes circulares

Biblioteca Math:
https://pypi.org/project/python-math/

[78]:
```
!pip install python-math
```

```
Collecting python-math
  Downloading python_math-0.0.1-py3-none-any.whl (2.4 kB)
Installing collected packages: python-math
Successfully installed python-math-0.0.1
```

[79]:
```
import math
```

[80]:
```
pi = math.pi
pi
```

[80]:
```
3.141592653589793
```

[81]:
```
h = float(input('Qual é a altura? Em metros. '))
d = float(input('Qual é o diâmetro? Em metros '))

area_l = h*2*(d/2)
```

```
area_b = 2*(pi*(d/2)**2)
area_t = round(area_l + area_b, 2)

valor = 580
valor_t = round(valor*area_t, 2)

print(f"A área total das chapas: {area_t}")
print('Preço total das chapas: R$ {}'.format(valor_t))
```

```
Qual é a altura? Em metros. 1
Qual é o diâmetro? Em metros 0.5
A área total das chapas: 0.89
Preço total das chapas: R$ 516.2
```

7.1.2 Exemplo 2 - Custos e ganhos de um funcionário

Receba o salário-base de um funcionário. Calcule e imprima o valor do salário-base considerando 7% de bônus de gratificação e o desconto de 7% de imposto sob estes ganhos.

[82]:
```
salario = float(input('Qual salário-base? Em R$ '))
ganhos_b = salario*1.07
ganhos_t = ganhos_b - 0.07*(ganhos_b)
print('O valor bruto a ser ganho é de R$ %.2f, enquanto seu valor líquido é de␣
 ↪R$ %.2f'%(ganhos_b, ganhos_t))
```

```
Qual salário-base? Em R$ 1200
O valor bruto a ser ganho é de R$ 1284.00, enquanto seu valor líquido é de R$
1194.12
```

8 DESAFIO I - Aquecimento de água em um tanque

Construa um programa que forneça o tamanho do diâmetro e altura de uma caldeira cilíndrica contendo água pura e imprima quanto de calor seria necessário para aquecê-la de 25 °C até 80ºC, desprezando efeitos dissipativos. Dados: calor específico da água = 1 cal/g ºC. e massa específica = 1000 kg/m³
Q = m.c.deltaT

[83]:
```
import math
pi = math.pi

d = float(input('Qual é o diâmetro da caldeira? Em metros.'))
h = float(input('Qual é a altura da caldeira? Em metros.'))

densidade = 1000
calor_s = 1
volume = round(h*(((d/2)**2)*pi), 2)
massa = round(densidade*volume, 2)
```

```
q = round(massa*calor_s*(80-25), 2)

print(f"Volume de água: {volume} m²")
print(f"Massa de água: {massa} kg")
print(f"Calor necessário: {q} cal")
```

```
Qual é o diâmetro da caldeira? Em metros.1
Qual é a altura da caldeira? Em metros.0.5
Volume de água: 0.39 m²
Massa de água: 390.0 kg
Calor necessário: 21450.0 cal
```

9 DESAFIO II - Salmoura em uma caldeira

Construa um programa que leia as dimensões de uma caldeira cilíndrica e a concentração de NaCl (em % m/m) necessária. Retorne o valor do volume da caldeira e a quantidade em kilogramas de NaCl. - Lembrando que a solução máxima do cloreto de sódio em água é de 36% em 25 ºC

```
massa_sal = Concentração(%)*massa_agua
```

[84]:
```
import math
pi = math.pi

d = float(input('Qual é o diâmetro da caldeira? Em metros.'))
h = float(input('Qual é a altura da caldeira? Em metros.'))
conc = float(input('Qual é a concentração do sal? Em % (m/m) '))


densidade = 1000
volume = h*(((d/2)**2)*pi)
massa = densidade*volume

massa_sal = conc*massa

print('='*40)
print('Massa de água total: %.2f'%massa)
print('Massa de cloreto de sódio: %.2f'%massa_sal)
print('='*40)
```

```
Qual é o diâmetro da caldeira? Em metros.1
Qual é a altura da caldeira? Em metros.0.5
Qual é a concentração do sal? Em % (m/m) 20
========================================
Massa de água total: 392.70
Massa de cloreto de sódio: 7853.98
========================================
```

Seção 2 - Estruturas de Seleção

March 7, 2022

1 Estruturas de decisão

Na construção de algorítimos é comum nos depararmos com uma situação onde se deve analisar um **conjunto de condições** para uma **tomada de ações**. Estruturas de decisão são estruturas que criam uma condição lógica para que um determinado comando seja realizado. Vamos aplicar abaixo um exemplo em um programa que avalia a classificação de idade de um indivíduo através da estrutura **if** e **else**:

Antes de começar, quero informar ao leitor que nesta segunda seção teremos o primeiro contato do uso das **bibliotecas** System-specific parameters and functions (**sys**), **math** e **numpy** para realizar alguns comandos ao decorrer dos exemplos. Este recurso julgo o mais importante, pois a linguagem Python não podemos perder tempo "inventando a roda", construindo algorítimos desnecessário. Dessa forma, utilizarei aos poucos diferentes bibliotecas aula-em-aula para evitar uma condução massante do conteúdo do curso.

BOA AULA!

[1]:
```
idade = int(input())
if idade < 18:
    print('Menor de idade')
else:
    print('Maior de idade')
```

```
18
Maior de idade
```

1.1 Exemplo 1: Ímpar ou Par

Construa um programa que leia um valor e determine se este é ímpar ou par.

[2]:
```
num = int(input())
if num % 2 == 0:
    print('Número par')
else:
    print('Número ímpar')
```

```
6
Número par
```

1.2 Exemplo 2: Avaliação de um aluno

1.2.1 Uso do "elif"

Construa um programa que leia as notas da P1, P2 e P3 e avalie se aluno está aprovado, recuperação ou reprovado. A P3 possui peso 2. - Aprovado Nf > 6 - Recuperação 5 <= Nf <= 6 - Reprovado Nf < 6

$Nf = \frac{(P1+P2+2P3)}{4}$

[3]:
```
p1 = float(input('Nota da prova 1: '))
p2 = float(input('Nota da prova 2: '))
p3 = float(input('Nota da prova 3: '))

media = (p1+p2+2*p3)/4

print('A média do aluno é de {}'.format(media))
if media > 6:
    print('Aprovado')
elif media >= 5 and media <=6:
    print('O aluno está de recuperação')
else:
    print('Reprovado')
```

```
Nota da prova 1: 4
Nota da prova 2: 5
Nota da prova 3: 8
A média do aluno é de 6.25
Aprovado
```

1.3 Exemplo 3: Resolução de Equação de segundo grau

Construa um programa para resolução de equação do segundo grau utilizando a fórmula de Bhaskara, onde indica ao usuário as seguintes mensagens.

Equação de Bhaskara

$$\Delta = b^2 - 4ac$$

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

$\Delta > 0$, há solução real

$\Delta = 0$, há solução real

$\Delta < 0, \quad$ não há solução real

[4]:
```
import sys
a = float(input('Coeficiente a: '))
if a == 0:
    print('Não se trata de uma equação de segundo grau')
    sys.exit()
b = float(input('Coeficiente b: '))
c = float(input('Coeficiente c: '))
delta = (b**2)-4*a*c
if delta<0:
    print('Não há uma solução real')
elif delta == 0:
    x = -b/2*a
    print('Há uma solução real, x = %.1f'%x)
else:
    x1 = (-b+(delta**0.5))/2*a
    x2 = (-b-(delta**0.5))/2*a
    print('Há duas soluções reais, x1 = %.1f e x2 = %.1f'%(x1,x2))
```

```
Coeficiente a: 1
Coeficiente b: 5
Coeficiente c: 3
Há duas soluções reais, x1 = -0.7 e x2 = -4.3
```

1.4 DESAFIO 1: Escala de pH

Construa um programa que recebe o valor da concentração de H3O+ e determine o seu valor de pH e se a suposta solução é ácida, básica ou neutra. Lembre-se que:

$pH = -log[H_3O^+]$

Utilize as bibliotecas **math** e **numpy**, caso achar necessário na resolução do exercício, pesquise na documentação utilizando os links oferecidos.

[5]:
```
import numpy as np
conc = float(input('Concentração de hidroxônio (mol/L):  '))

pH = -np.log10(conc)

if pH == 7:
    print('pH = %.2f, solução neutra.'%pH)
elif pH > 7:
    print('pH = %.2f, solução básica.'%pH)
else:
    print('pH = %.2f, solução ácida.'%pH)
```

```
Concentração de hidroxônio (mol/L):  2
pH = -0.30, solução ácida.
```

1.5 DESAFIO 2 - Salmoura em uma caldeira

Construa um programa que leia as dimensões de uma caldeira cilíndrica e a concentração de NaCl (em % m/m) necessária. Retorne o valor do volume da caldeira e a quantidade em kilogramas de NaCl. Caso a concentração de sal for superior à 36%, calcule e imprime o valor de sal precipitado na caldeira. - Lembrando que a solução máxima do cloreto de sódio em água é de 36% em 25 ºC

[6]:
```
import math
pi = math.pi

d = float(input('Qual é o diâmetro da caldeira? Em metros.'))
h = float(input('Qual é a altura da caldeira? Em metros.'))
conc = float(input('Qual é a concentração do sal? Em % (m/m) '))

conc /= 100
densidade = 1000
volume = h*(((d/2)**2)*pi)
massa = densidade*volume

massa_sal = conc*massa

if conc <0.36:
    print('='*40)
    print('Massa de água total: %.2f kg'%massa)
    print('Massa de cloreto de sódio em solução: %.2f kg'%massa_sal)
    print('='*40)
else:
    massa_ppt = (conc - 0.36)*massa
    print('='*40)
    print('Massa de água total: %.2f'%massa)
    print('Massa de cloreto de sódio em solução: %.2f kg'%massa_sal)
    print('Massa de cloreto de sódio precipitado: %.2f kg'%massa_ppt)
    print('='*40)
```

```
Qual é o diâmetro da caldeira? Em metros.1
Qual é a altura da caldeira? Em metros.5
Qual é a concentração do sal? Em % (m/m) 2
========================================
Massa de água total: 3926.99 kg
Massa de cloreto de sódio em solução: 78.54 kg
========================================
```

1.6 DESAFIO EXTRA: Cálculo de pH parte II

Construa um programa leia os valores da constante ácida e concetração molar, então calcule o pH considerando as seguintes condições:

- para ácidos fortes, em concentrações acima de 1E10-6 mol/l.
- para ácidos fracos, em concentrações acima de 1E10-6 mol/l.
- para ácidos fortes, em concetntrações abaixo de 1E10-6 mol/l (considerar efeito de autoprotólise).

Classifique:

$$Ka > 1 \quad \text{para ácidos fortes}$$

$$Ka < 1 \quad \text{para ácidos fracos}$$

Dados do Problema

$pH = -log[H_3O^+]$
$k_w = 10^{-14}$
$[H_3O^+]^2 - [HCl]_{inicial}[H_3O^+] - K_w = 0$
$[H_3O^+] \approx \sqrt{[HA]K_a}, \quad \text{assumindo} \quad [HA] >> [H_3O^+]$

[9]:
```
import math  #importação da biblioteca math
ka = float(input('Constante ácida (Ka): '))  #leitura de dados utilzando o␣
 ↪comando input
conc = float(input('Concentração, em mol/L: '))  #leitura de dados utilzando o␣
 ↪comando input
if ka > 1:  #ka>1 engloba ácidos fortes
    if conc >= 1e-6:  #1e-6 é maior que 0, é desconsiderado efeitos de protólise
        pH = -math.log(conc,10)  #resolução do desafio 1
        print(f'Ácido forte sem efeitos de autoprotólise, pH =  %.3f'%pH) ␣
 ↪#resultado para ocasião 1
    else:  #1e-6 é menor que 0, portanto, considera-se efeitos de protólise
        a = 1
        b = -conc
        c = -1e-14
    if a == 0:
        print('Equação de segundo grau inexistente')
    else:
        delta = (b**2)-(4*a*c)
        if delta > 0:
            x1 = (-b+(delta**0.5))/(2*a)
            x2 = (-b-(delta**0.5))/(2*a)
        if x1>=0:  #esta estrutura de decisão seleciona somente valores␣
 ↪positivos
            pH = -1*math.log(x1,10)
            print(f'Ácido forte com efeitos de autoprotólise, pH =  %.3f'%pH) ␣
 ↪#resultado para ocasião 2
```

```
        elif x2>=0:
            pH = -1*math.log(x2,10)
            print(f'Ácido forte com efeitos de autoprotólise, pH =  %.3f'%pH) ␣
↪#resultado para ocasião 2
        else:
            print('Não há resolução')
        if delta == 0:
            x1 = -b/2*a
        if x1>=0: #esta estrutura de decisão seleciona somente valores positivos
            pH = -1*math.log(x1,10)
            print(f'Ácido forte com efeitos de autoprotólise3, pH =  %.3f'%pH) ␣
↪#resultado para ocasião 2
        else:
            print('Não há resolução')
else:  #ka<1 engloba ácidos fracos
    h30 = (conc*ka)**0.5
    pH = round(-1*math.log(h30,10),2)
    print(f'Ácido fraco sem efeitos de autoprotólise, %.3f'%pH)  #resultado␣
↪para ocasião 3
```

```
Constante ácida (Ka): 1
Concentração, em mol/L: 2
Ácido fraco sem efeitos de autoprotólise, -0.150
```

2 Links Úteis

Segue os links com as bibliotecas utilizadas nesta aula. O uso deste tipo de marterial é essencial, uma vez que você poderá explorar todos recursos disponíveis da biblioteca. Aqui no curso de Python Básico apresentarei os principais comandos, infelizmente, no entanto, não será possível apresentar tudo que elas podem oferecer. Com isso, espero que seja habitual o uso dos links de documentações quando houver dúvidas.

SYSTEM-SPECIFIC PARAMETER AND FUNCTIONS (sys):
https://docs.python.org/3/library/sys.html
MATH:
https://docs.python.org/3/library/math.html
NUMPY:
https://numpy.org/doc/stable/user/index.html#user
PANDAS:
https://pandas.pydata.org/docs/user_guide/10min.html

Seção 3 - Estruturas de Repetição

March 7, 2022

1 COMANDO WHILE

Responsável de executar um conjuntos de comandos, o fim do loop se encerrará quando uma condição predeterminada ser atendida. Este tipo de comando nos permite realizar um conjunto de comandos inúmeras vezes, de modo que podemos coletar os dados do usuário para manter esse sistema de repetição. Nesta seção veremos as aplicações da estrutura while para criação de sequências numéricas e para coleta de dados continuada para realização de tarefas específicas.

1.1 Sintaxe de condição numérica

Forma de utilizar o comando while de modo que é predeterminado um número específico de execuções do loop.

1.1.1 Exemplo 1: Sequências númericas:

Construa um programa que leia o número de termos das sequência abaixo e retorne respectivamente com o valor de sua soma.

$$1 + 2 + 3 + 4 + 5...$$

$$1 - 2 + 3 - 4 + 5...$$

$$1 - 3 + 5 - 7 + 9...$$

[4]:
```
termos = int(input())
serie1 = list()
i = 0
while (i<termos):
    i += 1
    serie1.append(i)
print(serie1)
```

```
20
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Note que neste primeiro exemplo precisamos de um contador `i` para definir a condição numérica e dentro da estrutura while foi inserido a operação `i += 1` para gerar a sequência numérica. Na segunda sequência, precisa-se de um novo contador para ser o expoente na sequência alternada, observe que a soma do contador `n` foi escrito na última linha do código.

[12]:
```
termos = int(input())
serie2 = list()
i = 0
n = 0
while (i<termos):
    i += 1
    serie2.append(i*(-1)**n)
    n += 1
print(serie2)
```

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

Agora vamos aplicar o mesmo raciocínio, só que neste momento vamos selecionar os valores ímpares. Durante o vídeo eu demonstrei um raciocínio utilizando a equação `I = 2n + 1`, porém aqui nesta apostila de acompanhamento deixo o código mais enxuto utilizando um comando de seleção.

[14]:
```
termos = int(input())
serie3 = list()
i = 0
n = 0
while (i<termos):
    i += 1
    if (i % 2 == 1):
        serie3.append(i*(-1)**n)
        n += 1
print(serie3)
```

```
20
[1, -3, 5, -7, 9, -11, 13, -15, 17, -19]
```

1.2 Sintaxe de condição lógica

Neste caso não há um número predeterminado de execuções do loop, dessa maneira, precisa-se inserir o comando 'break' quando a condição lógica for atendida.

1.2.1 Exemplo 2: lista de produtos do mercado

Utiliza-se comumente estruturas `while True` para criar um algorimo que receba entradas, de modo que, ao finalizar todos os comandos do laço, o programa interage com usuário peguntando se ele deseja continuar. Veja o exemplo clássico sobre o carrinho de uma loja genérica abaixo:

[15]:
```
produtos = []
while True:
    produto = input('Mercadoria: ')
```

```
    resposta = input('Adicionar mais um item a lista? Digite "n" para␣
↪finalizar')
    produtos.append(produto)
    if (resposta.lower() == 'n'):
        print('Operação finalizada')
        print('Lista de produtos:  {}'.format(produtos))
        break
```

```
Mercadoria: Banana
Adicionar mais um item a lista? Digite "n" para finalizar
Mercadoria: Feijão
Adicionar mais um item a lista? Digite "n" para finalizara
Mercadoria: Arroz
Adicionar mais um item a lista? Digite "n" para finalizar
Mercadoria: Batata
Adicionar mais um item a lista? Digite "n" para finalizar
Mercadoria: Mousse de Limão
Adicionar mais um item a lista? Digite "n" para finalizarn
Operação finalizada
Lista de produtos:  ['Banana', 'Feijão', 'Arroz', 'Batata', 'Mousse de Limão']
```

1.2.2 Exemplo 3: Cálculo de calor em caldeira - Parte 2

Construa um programa que calcula o valor de calor de caldeiras cilínndricas e depois adicione em uma lista com os valores organizados de maneira decrescente.

[16]:
```
import math
pi = math.pi
while True:
    d = float(input('Qual é o diâmetro da caldeira? Em metros.'))
    h = float(input('Qual é a altura da caldeira? Em metros.'))

    densidade = 1000
    calor_s = 1
    volume = round(h*(((d/2)**2)*pi), 2)
    massa = round(densidade*volume, 2)

    q = round(massa*calor_s*(80-25), 2)

    print(f"Volume de água: {volume} m²")
    print(f"Massa de água: {massa} kg")
    print(f"Calor necessário: {q} cal")
    resposta = input('Calcular outra unidade? Digite "N" para sair')
    if resposta.upper() == 'N':
        print('Operação finalizada')
        break
```

```
Qual é o diâmetro da caldeira? Em metros.0.7
Qual é a altura da caldeira? Em metros.0.8
```

```
Volume de água: 0.31 m²
Massa de água: 310.0 kg
Calor necessário: 17050.0 cal
Calcular outra unidade? Digite "N" para sair
Qual é o diâmetro da caldeira? Em metros.0.5
Qual é a altura da caldeira? Em metros.0.6
Volume de água: 0.12 m²
Massa de água: 120.0 kg
Calor necessário: 6600.0 cal
Calcular outra unidade? Digite "N" para sair
Qual é o diâmetro da caldeira? Em metros.1
Qual é a altura da caldeira? Em metros.2
Volume de água: 1.57 m²
Massa de água: 1570.0 kg
Calor necessário: 86350.0 cal
Calcular outra unidade? Digite "N" para sairn
Operação finalizada
```

1.2.3 Exemplo 4: Registro de dados de um aluno - Parte 2

Construa um programa que retorna um formulario com os dados de cada aluno em um dicionario. Cada ficha do aluno deve conter o nome, p1, p2, p3, média e condição de aprovação.

[38]:
```python
# Exemplo 2 - Seção 2 - Estruturas de decisão
formulario = list()
while True:
    ficha = dict()
    nome = input('Nome do Aluno:  ')
    p1 = float(input('Nota da prova 1: '))
    p2 = float(input('Nota da prova 2: '))
    p3 = float(input('Nota da prova 3: '))

    media = (p1+p2+p3)/3

    ficha['Nome'] = nome
    ficha['Prova 1'] = p1
    ficha['Prova 2'] = p2
    ficha['Prova 3'] = p3
    ficha['Média'] = media

    if media > 6:
        print('Aprovado')
        ficha['Status'] = 'Aprovado'
    elif media >= 5 and media <=6:
        print('O aluno está de recuperação')
        ficha['Status'] = 'Recuperação'
    else:
        print('Reprovado')
```

```
        ficha['Status'] = 'Reprovação'
    formulario.append(ficha)
    resposta = input('Registrar mais um aluno? Digite "n" para finalizar')

    if resposta.lower() == 'n':
        print('Operação finalizada')
        break
```

```
Nome do Aluno:  André
Nota da prova 1: 6
Nota da prova 2: 5.9
Nota da prova 3: 6.1
O aluno está de recuperação
Registrar mais um aluno? Digite "n" para finalizar
Nome do Aluno:  Neri
Nota da prova 1: 7
Nota da prova 2: 8
Nota da prova 3: 5
Aprovado
Registrar mais um aluno? Digite "n" para finalizar
Nome do Aluno:  Pedro
Nota da prova 1: 3
Nota da prova 2: 4
Nota da prova 3: 10
O aluno está de recuperação
Registrar mais um aluno? Digite "n" para finalizar
Nome do Aluno:  Fernanda
Nota da prova 1: 8
Nota da prova 2: 8
Nota da prova 3: 7
Aprovado
Registrar mais um aluno? Digite "n" para finalizarn
Operação finalizada
```

[39]:
```
formulario
```

[39]:
```
[{'Nome': 'André',
  'Prova 1': 6.0,
  'Prova 2': 5.9,
  'Prova 3': 6.1,
  'Média': 6.0,
  'Status': 'Recuperação'},
 {'Nome': 'Neri',
  'Prova 1': 7.0,
  'Prova 2': 8.0,
  'Prova 3': 5.0,
  'Média': 6.666666666666667,
  'Status': 'Aprovado'},
```

```
 {'Nome': 'Pedro',
  'Prova 1': 3.0,
  'Prova 2': 4.0,
  'Prova 3': 10.0,
  'Média': 5.666666666666667,
  'Status': 'Recuperação'},
 {'Nome': 'Fernanda',
  'Prova 1': 8.0,
  'Prova 2': 8.0,
  'Prova 3': 7.0,
  'Média': 7.666666666666667,
  'Status': 'Aprovado'}]
```

2 Desafio 1: Dados experimentais

Construa um programa que recolha um-a-um dos dados experimentais de tempo e concentração de uma amostra. depois, contrua um gráfico linear ln[C3H6] x tempo. (Utilize o comando while)

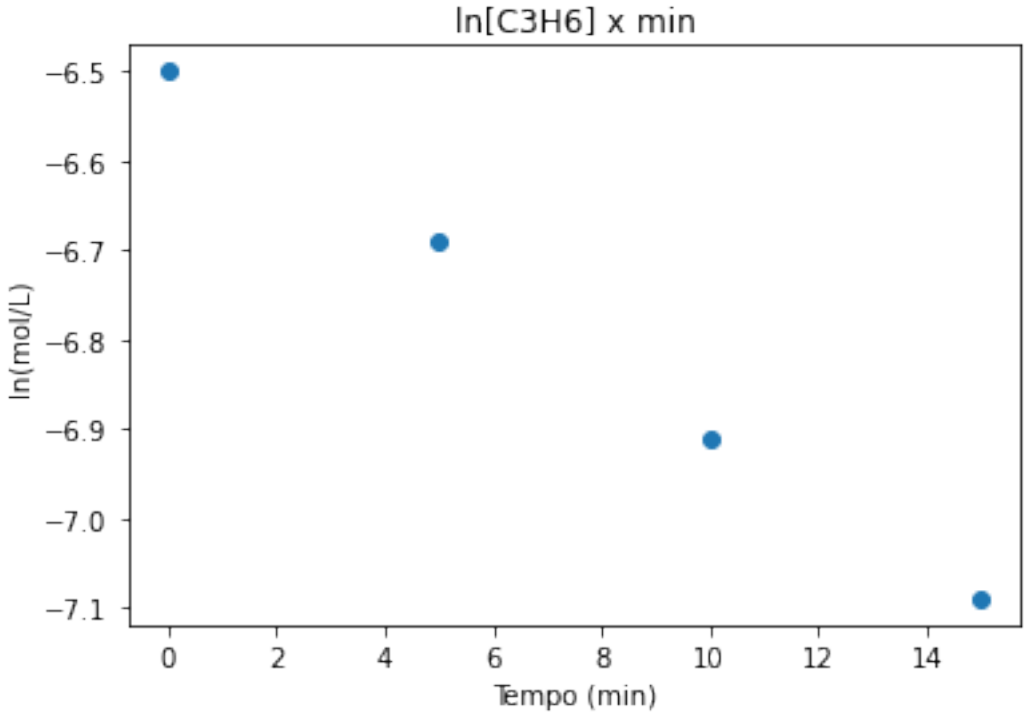
___Utilize o link abaixo para acessar o tutorial sobre o gráfico _scatter Real Python___:
https://realpython.com/visualizing-python-plt-scatter/

MÃOS À OBRA!

[23]:
```
concentracao, tempo = [], []

while True:
    dados1 = float(input('Concentração, em mol/L: '))
    dados2 = float(input('Tempo, em min: '))
    concentracao.append(dados1)
    tempo.append(dados2)

    resposta = input('Finalizar coleta dados? Digite "s" para finalizar')

    if resposta.upper() == 'S':
        break

ln =[]
n=0
import math
while n<len(concentracao) :
    ln.append(round(math.log(concentracao[n]),2))
    n += 1

import matplotlib.pyplot as plt
```

```
plt.scatter(tempo,ln)
plt.xlabel('Tempo (min)')
plt.ylabel('ln(mol/L)')
plt.title('ln[C3H6] x min')
plt.show()
```

```
Concentração, em mol/L: 1.5E-3
Tempo, em min: 0
Finalizar coleta dados? Digite "s" para finalizar
Concentração, em mol/L: 1.24E-3
Tempo, em min: 5
Finalizar coleta dados? Digite "s" para finalizar
Concentração, em mol/L: 1E-3
Tempo, em min: 10
Finalizar coleta dados? Digite "s" para finalizar
Concentração, em mol/L: 0.83E-3
Tempo, em min: 15
Finalizar coleta dados? Digite "s" para finalizars
```



[24]:
```
import numpy as np
ln1 = np.array(ln)
type(ln1)
```

[24]: numpy.ndarray

[25]:
```
np.log(np.array(concentracao))
```

[25]:
```
array([-6.50229017, -6.6926439 , -6.90775528, -7.09408486])
```

3 RANGE

Esta função retorna uma sequência numérica que é definida através do argumento da função.
Ela pode ser utilizada de três maneiras:

- **Stop possui intervalo aberto;**
- **Start possui intervalo fechado;**
- **Por padrão, step = 1;**
- **É possível construir sequências negativas.**

A função range será um ótimo auxiliar para criarmos a estrtura `for`, uma vez que a criação de coleções é a essencia deste tipo de estrutura de repetição.

3.0.1 Método 1 - range(stop)

[1]:
```
list(range(10))
```

[1]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

3.0.2 Método 2 - range(start,stop)

[26]:
```
list(range(1,11))
```

[26]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

3.0.3 Método 3 - range(start,stop,step)

[3]:
```
list(range(1,22,2))
```

[3]:
```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
```

3.1 Desafio em aula

Construa um programa que leia um valor de um intervalor contendo números naturais e depois realize as seguintes operações: - Soma total, media e desvio padrão de todos os números ímpares - Soma total, media e desvio padrão de todos os números pares - Soma total, media e desvio padrão de todos os números multíplos de 7 e 11.

Dica: Utilize a biblioteca numpy para resolução do terceiro item
Tutorial da Geekforgeeks.org sobre medidas de tendência central e de dispersão com Numpy:
https://www.geeksforgeeks.org/describe-a-numpy-array-in-python/#:~:text=NumPy%20is%20a%20Python%20li

[27]:
```
print(
list(range(1,100,2))
) # I = 2n + 1
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41,
43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81,
83, 85, 87, 89, 91, 93, 95, 97, 99]
```

[28]:
```
print(
list(range(0,100,2))
)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40,
42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80,
82, 84, 86, 88, 90, 92, 94, 96, 98]
```

[29]:
```
lista = list(range(0,100,7)) + list(range(0,100,11))
#lista.sort() # Ordena de forma de crescente os itens de uma lista
import numpy as np
array = np.array(lista)
print('Média: {}'.format(array.mean()))
print('Soma: {}'.format(array.sum()))
print('Desvio Padrão: {0:.1f}'.format(array.std()))
```

```
Média: 49.2
Soma: 1230
Desvio Padrão: 30.8
```

4 COMANDO FOR

Responsável de executar um conjuntos de comandos, o fim do loop será dado por um número limitado de vezes.

4.0.1 Exemplo 1: Impressão de itens de uma lista

Imprima os nomes da lista abaixo:

```
alunos = ['André', 'Maria', 'João', 'Luis', 'Neri', 'Fernanda', 'Pedro', 'Rita', 'Gabriel']
```

[30]:
```
alunos = ['André', 'Maria', 'João', 'Luis', 'Neri', 'Fernanda', 'Pedro',␣
↪'Rita', 'Gabriel']
```

[31]:
```
for nomes in alunos:
    print(nomes)
```

```
André
Maria
João
Luis
Neri
```

```
Fernanda
Pedro
Rita
Gabriel
```

4.0.2 Exemplo 2: Cálculo do fatorial de um valor inteiro

Construa um programa que recebe um valor inteiro e retorne o valor do seu fatorial
$5! = 5x4x3x2x1 = 120$

[35]:
```
num = int(input('Número de termos: '))
fatorial = 1
for i in range(1,num+1):
    fatorial *= i
print('{}!={}'.format(num,fatorial))
```

```
Número de termos: 10
10!=3628800
```

4.0.3 Exemplo 3: Soma dos quadrados x O quadrado da soma

Leia o número de termos e depois calcule a diferença entre a soma dos quadrados e o quadrado da soma de valores naturais

Soma dos quadrados:

$1^2 + 2^2 + 3^2 + 4^2 + 5^2...$

Quadrado da soma:

$(1 + 2 + 3 + 4 + 5)^2$

[36]:
```
termos = int(input('Termos: '))

soma_q, quad_s = [], []

for i in range(1, termos+1):
    soma_q.append(i**2)
    quad_s.append(i)
dif = sum(soma_q) - sum(quad_s)
print(f"Diferença entre as séries: {dif}")
```

```
Termos: 10
Diferença entre as séries: 330
```

4.0.4 Exemplo 4: Aplicação de For Aninhado - Relógio

```
import time
for hora in range(60):
    for minuto in range(60):
        for segundo in range(60):
```

```
            print('{}:{}:{}'.format(hora,minuto,segundo))
            time.sleep(1)
```

5 Desafio 2 - Investimento de renda fixa

Exercício adaptado de Vanessa Braganholo
http://www2.ic.uff.br/~vanessa/material/prog-python/05-Repeticao.pdf

Faça um programa que calcule o retorno de investimento com 6% de juros compostos com as seguintes características.

- **Retorne o saldo atual do mês**
- **Retorne o saldo anual levando em consideração o desconto de 7% do valor bruto.**
- **Pergunte ao usuário se deseja manter o investimento com o ganho de 0.5% acrescentados ao juros.**
- **Ao retirar o investimento pague mais 15% de taxa**
  — **Exemplificação:**

```
Mês 1 :       R$ 1200.00
Mês 2 :       R$ 1272,00
...
Mês 12 :       R$ 2.414,63
Saldo final:   R$ 2,245.61
Deseja continuar com o nosso plano? [S/N]
```

[46]:
```
p = float(input('Valor inicial:  '))
saldo = list()
juros = 0.06

while True:

    for i in range(1,13):
        juros_compostos = (juros+1)**i
        montante = juros_compostos*p
        saldo.append(montante)
        print('Saldo mensal: {0:.2f}'.format(montante))

    saldo_anual = saldo[-1]*0.93

    print('Saldo Anual: {0:.2f}'.format(saldo_anual))

    juros += 0.005
    p = saldo_anual
    resposta = input('Quer continuar? Digite "n" para encerrar o investimento.')

    if resposta.lower() == 'n':
        print('Operação Finalizada')
```

```
        print('O saldo depositado na conta do cliente: {0:.2f}'.
↪format(saldo_anual*0.85))
        p=0
        break
```

```
Valor inicial:  1200
Saldo mensal: 1272.00
Saldo mensal: 1348.32
Saldo mensal: 1429.22
Saldo mensal: 1514.97
Saldo mensal: 1605.87
Saldo mensal: 1702.22
Saldo mensal: 1804.36
Saldo mensal: 1912.62
Saldo mensal: 2027.37
Saldo mensal: 2149.02
Saldo mensal: 2277.96
Saldo mensal: 2414.64
Saldo Anual: 2245.61
Quer continuar? Digite "n" para encerrar o investimento.a
Saldo mensal: 2391.58
Saldo mensal: 2547.03
Saldo mensal: 2712.59
Saldo mensal: 2888.90
Saldo mensal: 3076.68
Saldo mensal: 3276.67
Saldo mensal: 3489.65
Saldo mensal: 3716.48
Saldo mensal: 3958.05
Saldo mensal: 4215.32
Saldo mensal: 4489.32
Saldo mensal: 4781.12
Saldo Anual: 4446.44
Quer continuar? Digite "n" para encerrar o investimento.a
Saldo mensal: 4757.69
Saldo mensal: 5090.73
Saldo mensal: 5447.09
Saldo mensal: 5828.38
Saldo mensal: 6236.37
Saldo mensal: 6672.91
Saldo mensal: 7140.02
Saldo mensal: 7639.82
Saldo mensal: 8174.61
Saldo mensal: 8746.83
Saldo mensal: 9359.11
Saldo mensal: 10014.24
Saldo Anual: 9313.25
Quer continuar? Digite "n" para encerrar o investimento.n
```

```
Operação Finalizada
O saldo depositado na conta do cliente: 7916.259571174125
```

6 Desafio 3 - Série de Taylor

Faça um programa que imprima uma lista com n termo da sequencia abaixo juntamente com a sua soma total.
(Utilize o comando while)

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$$

Observação: - Será recebido o valor de n e deve ser atribuído x=2. - Utilize o método factorial() **da math library - Tente resolver sem a biblioteca Numpy**

[6]:
```
import math
termos = int(input('Número de termos: '))
x = int(input())

# fatorial
l_fatorial =list()
for fatorial in range(termos+1):
    l_fatorial.append((math.factorial(fatorial)))

# x
l_x = [ ]
for n in range(termos+1):
    l_x.append(x**n)

#taylor
taylor=[]
for i in range(len(l_x)):
    taylor.append(l_x[i]/l_fatorial[i])
print('Soma da série Taylor:  {}'.format(sum(taylor)))
```

```
Número de termos: 1000
1
Soma da série Taylor:  2.7182818284590455
```

[7]:
```
math.e
```

[7]:
```
2.718281828459045
```

Seção 4 - Funções e Módulos

March 7, 2022

1 Funções

Trata-se de um conjunto de comandos que é inserido dentro de um bloco de código que possui um nome e, opcionalmente, definidos parâmetros para sua definição. Por questões de **boas práticas de programação**, os nomes de funções são **verbos de ação** e em **caixa baixa**, por exemplo, uma função que imprime o nome deste treinamento "Curso de Python Básico", então o nome adequado para tal seria **imprimir_nome**

1.1 Funções Simples

Funções simples são muito utilizadas para compactar códigos que não precisam de um retorno específico e que não recebam nenhum parâmetro.

1.1.1 Exemplo 1 - Imprimir nome

Construa uma função que imprime o nome deste curso.

[3]:
```
def imprimir_nome():
    print('Curso de Python Básico')
imprimir_nome()
```

```
Curso de Python Básico
```

1.2 Funções com Parâmetros e Retorno

- O(s) **Parâmetro(s)** de uma função são as informações que recebemos em uma função para realizar uma determinada tarefa.
- O **retorno** é responsável de enviar o dado(s) desejado(s) gerado dentro da função, a partir daqui construiremos funções somente com retorno.

Na seção 1, na aula sobre função input() resolvemos exercícios onde recebiámos as dimensões de um cilindro reto para calcular um determinado preço do aço inox para sua construção, vejamos isto dentro de uma função.

1.2.1 Exemplo 2 - Refaroração de Código

Exemplo 1 (Seção 1) - Construção de tanque cilídrico inox

Construa um programa que leia diâmetro e altura de um cilíndro e retorna o usuário valor e metragem das chapas inox necessárias para construção do cilindro.

- Preço por m² da chapa: R$580,00
- A empresa realiza cortes circulares

TRECHO DE CÓDIGO

```
import math
h = float(input('Qual é a altura? Em metros. '))
d = float(input('Qual é o diãmetro? Em metros '))
area_l = h*2*(d/2)
area_b = 2*(math.pi*(d/2)**2)
area_t = round(area_l + area_b, 2)
valor = 580
valor_t = round(valor*area_t, 2)
print(f"A área total das chapas: {area_t}")
print('Preço total das chapas: R$ {}'.format(valor_t))
```

[8]:
```
import math
def calcular_preco(altura,diametro):

    area_l = altura*2*(diametro/2)
    area_b = 2*(math.pi*(diametro/2)**2)
    area_t = round(area_l + area_b, 2)
    valor = 580
    valor_t = round(valor*area_t, 2)

    print(f"A área total das chapas: {area_t}")
    print('Preço total das chapas: R$ {}'.format(valor_t))

    return [valor_t,area_t]
```

[10]:
```
calcular_preco(1,0.5)
```

```
A área total das chapas: 0.89
Preço total das chapas: R$ 516.2
```

[10]:
```
[516.2, 0.89]
```

1.3 Função dentro de outra função e Parâmetros fixos

Em programação existe um conceito chamado **SOLYD**, trata-se de um conjunto de princípios de **estruturas padrões** que definem **boas práticas de codificação**.
E um dos conceitos é **não "sobrecarregar"** as tarefas de uma função, classe ou método. Para resolver isso, podemos inserir funções dentro de outras funções para **dividir o código** tornando mais **legível**.

1.3.1 Exemplo 3 - Parâmetro fixo - Equação linear

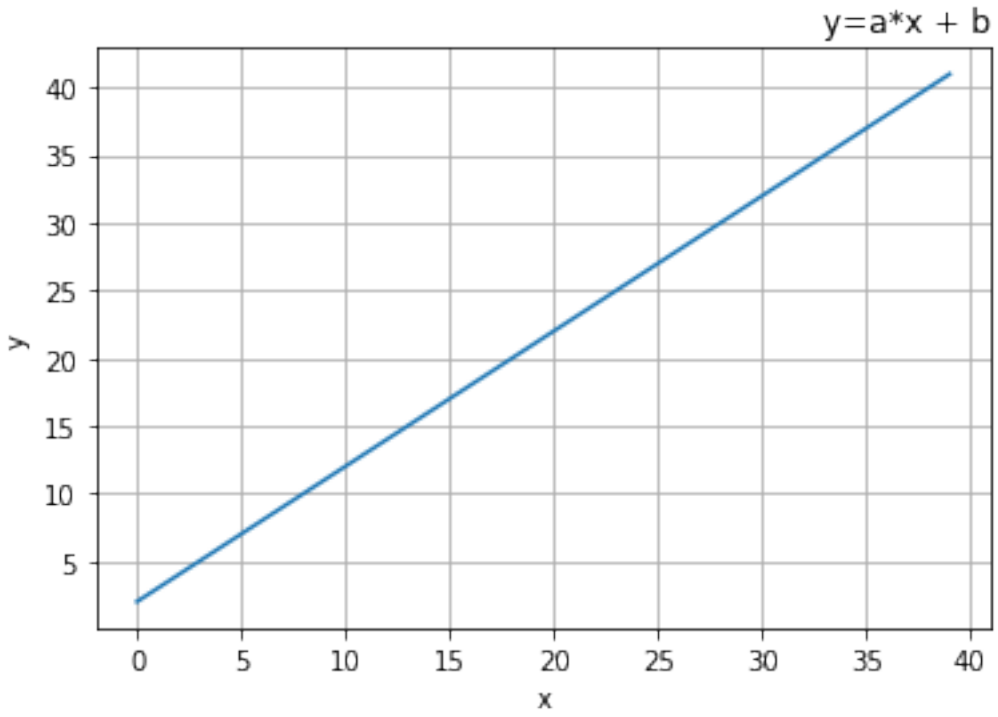
[45]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[69]:
```
def equacao_linear(a,b):

    x= np.array(range(40))
    plt.plot(x,a*x +b)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.title('y=a*x + b',, loc='right')
    return plt.show()
equacao_linear(1,5)
```

1.3.2 Exemplo 4 - Encapsulamento de função - Equação linear

Ainda no exemplo anterior, encapsule os comandos da edição do gráfico.

[73]:
```
def editar_grafico():
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.title('y=a*x + b', loc='right')

def equacao_linear2(a,b):
    x= np.array(range(40))
    plt.plot(x,a*x +b)
    editar_grafico()
    return plt.show()

equacao_linear2(1,2)
```



[83]:
```
sistema_caldeira()
```

```
Qual é o diâmetro da caldeira? Em metros.1
Qual é a altura da caldeira? Em metros.1
Variação de temperatura? Em celsius.50
Volume de água: 0.79 m²
Massa de água: 790.0 kg
Calor necessário: 39500.0 cal
Calcular outra unidade? Digite "N" para sairn
Operação finalizada
```

2 Documentando uma função através de Docstrings

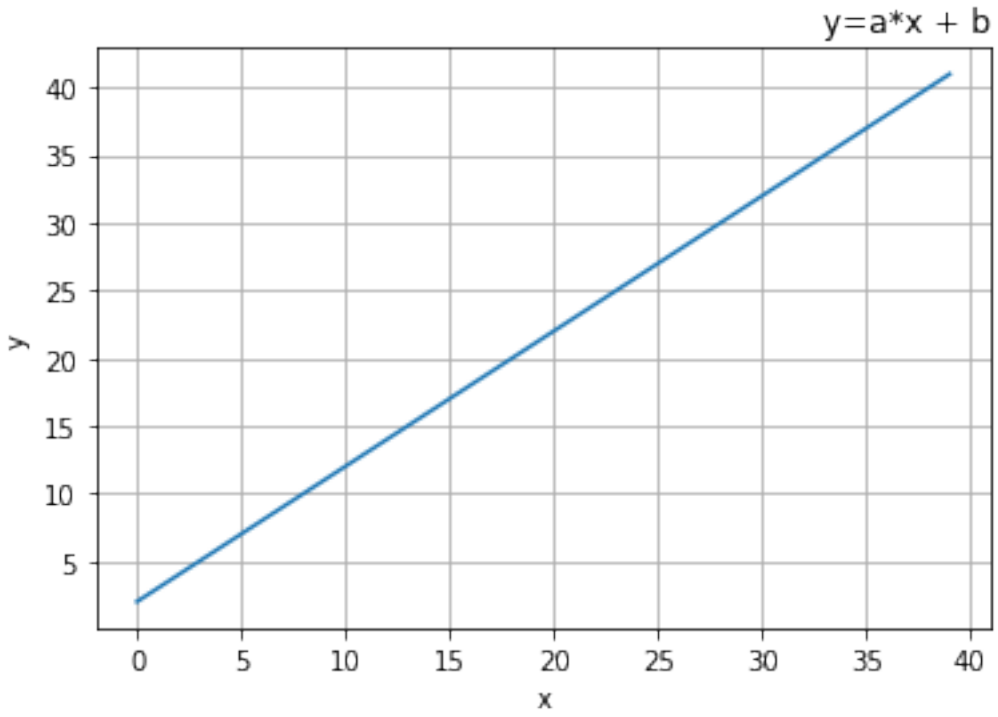
Documentar suas funções, métodos e classes é importante para a **manuntenção** na construção de um projeto, principalmente quando este código está sendo trabalhado em equipe, onde é preciso **clareza das informações** entre membros da equipe.

A documentação de uma função possui um determinado padrão de apresentação, onde se deve apresentar um breve resumo de sua funcionalidade, seguido dos parâmetros e retorno. Segue o exemplo:

[87]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(…)
    print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

[88]:
```python
def editar_grafico():
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.title('y=a*x + b', loc='right')

def equacao_linear2(a,b):
    """
    Função que recebe coeficientes de equação de 1 grau e retorna seu␣
 ↪respectivo gráfico linear
    a: Coeficiente angular
    b: Coeficiente linear
    return: Gráfico linear
    """
    x= np.array(range(40))
    plt.plot(x,a*x +b)
    editar_grafico()
    return plt.show()

equacao_linear2(1,2)
```



3 DESAFIO 1 - Refaroração de Código II

Tente encapsular ao máximo o código abaixo. Esta função deve receber a variação de temperatura em celsius também.

Dica:
- Crie 4 funções **Desafio 1 (Seção 1) - Calor de uma caldeira**

Construa um programa que forneça o tamanho do diâmetro e altura de uma caldeira cilíndrica contendo água pura e imprima quanto de calor seria necessário para aquecê-la de 25 °C até 80ºC, desprezando efeitos dissipativos. Dados: calor específico da água = 1 cal/g ºC. e massa específica = 1000 kg/m³.

$$Q = mc\Delta T$$

TRECHO DE CÓDIGO

```
import math
while True:
    d = float(input('Qual é o diâmetro da caldeira? Em metros.'))
    h = float(input('Qual é a altura da caldeira? Em metros.'))
```

```
    densidade = 1000
    calor_s = 1
    volume = round(h*(((d/2)**2)*pi), 2)
    massa = round(densidade*volume, 2)

    q = round(massa*calor_s*(80-25), 2)

    print(f"Volume de água: {volume} m²")
    print(f"Massa de água: {massa} kg")
    print(f"Calor necessário: {q} cal")
    resposta = input('Calcular outra unidade? Digite "N" para sair')
    if resposta.upper() == 'N':
        print('Operação finalizada')
```

[82]:
```
def definir_dimensoes():
    d = float(input('Qual é o diâmetro da caldeira? Em metros.'))
    h = float(input('Qual é a altura da caldeira? Em metros.'))
    deltaT = float(input('Variação de temperatura? Em celsius.'))
    return d,h,deltaT

def calcular_massa_volume(d,h,densidade=1000,calor_s=1):
    volume = round(h*(((d/2)**2)*math.pi), 2)
    massa = round(densidade*volume, 2)
    return massa, volume

def calcular_calor(d,h,␣
 ↪massa=calcular_massa_volume,calor_s=1,deltaT=definir_dimensoes()[2]):
    return round(massa(d,h)[0]*calor_s*(deltaT), 2)

def sistema_caldeira():
    """
    Calcula o calor necessário para aquecer à água em uma caldeira cilíndrica
    : return: Volume (m³) e massa (kg) de água contido na caldeira e calor␣
 ↪(calorias) necessário para o aquecimento
    """

    while True:
        d,h,deltaT = definir_dimensoes()
        print(f"Volume de água: {calcular_massa_volume(d,h)[1]} m²")
        print(f"Massa de água: {calcular_massa_volume(d,h)[0]} kg")
        print(f"Calor necessário: {calcular_calor(d,h)} cal")
        resposta = input('Calcular outra unidade? Digite "N" para sair')
        if resposta.upper() == 'N':
            return print('Operação finalizada')
```

```
Qual é o diâmetro da caldeira? Em metros.1
Qual é a altura da caldeira? Em metros.1
```

```
Variação de temperatura? Em celsius.50
```

4 Módulos

O módulo é um arquivo que contém comandos e definições em Python e que pode ser utilizados em outros programas.

Há módulos dentro de **pacotes internos do Python 3** e também e **bibliotecas externas**, como o Numpy, Matplotlib, Math, Pandas entre outros. Além disso, podemos criar os nossos módulos contendo próprias definições.

Nesta aula contruiremos um módulo contendo as funções de exemplo e desafio.

[1]:
```python
# Exemplo de módulo de bloco de código que deve conter o arquivo .py
import matplotlib.pyplot as plt
import numpy as np

def __editar_grafico():
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('y =ax +b', loc='right')

def plotar_grafico(a,b):
    """
    Função que plota um gráfico linear a partir dos valores dos coeficientes da␣
 ↪reta
    a: coeficiente angular
    b: coeficiente linear
    return: Gráfico plt.plot da equação polinomial do primeiro grau
    """
    x = np.array(range(20))
    plt.plot(x, a*x + b)
    __editar_grafico()
    return plt.show()
```

Seção 5 - Variáveis Compostas - Aplicação Química

March 7, 2022

1 Variáveis Compostas e Arquivos

1.1 Aplicação em Titulação Potenciométrica

Vamos mudar a **dinâmica** de nossas aulas, pois neste momento temos o domínio dos **conceitos básicos da linguagem** e agora abordaremos as **variáveis compostas** em **aplicações reais em química**. Vamos conduzir a aula como uma resolução prática da **Determinação da porcentagem de ácido acético em vinagre**, para isso, importaremos dados através da **interação do Excel com o Python**.

Todos os dados experimentais que utilizaremos nesta seção foram dados que eu obtive em minha experiência na graduação na disciplina de Laboratório de Análitica Clássica. No final desta seção criaremos um módulo contendo as definições para a aplicação da **Titulação Potenciométrica**.

O objetivo principal desta seção é criar um **treinamento** para a aplicação de **problemas do cotidiano de laboratório** utilizando os recursos da **linguagem do Python**, de modo que o usuário siga a lógica aplicada no exemplo abordado.

No meu Google Drive, deixarei dados experimentais de outras práticas para utilizar como treinamento.

Link:
https://drive.google.com/drive/folders/1yxZyj3doQOAvfH0O-6JgefvqxULgJ3vE?usp=sharing

2 1º Passo: Importação de dados com a biblioteca Pandas

Vamos importar o arquivo xlsx contendo os valores de pH e volume de solução padronizada de hidróxido de sódio. Baixe o arquivo **"titulacao_potenciometrica.xlsx"**.

Link:
https://docs.google.com/spreadsheets/d/11Q9tUWoWDxbBrqtIONxxpbotEdKDz5kt/edit?usp=sharing&ouid=1

Uma vez feito o *upload* do arquivo no diretório, utilizaremos o método **pd.read_excel("nome_do_arquivo.xlsx")**, dessa forma será gerado um **dataframe** do arquivo. Também é possível importar dados em arquivos com a extensão **CSV**, para isso, iremos usar o método **pd.read_csv("nome_do_arquivo.csv", sep=';')**.

[3]:
```
import pandas as pd
```

[4]:
```
def impotar_excel(nome):
    return pd.read_excel(nome)
impotar_excel('titulacao_potenciometrica.xlsx').head()
```

[4]:

	V(mL)	pH
0	0.0	3.40
1	1.0	3.65
2	2.0	3.93
3	3.0	4.12
4	4.0	4.32

[5]:
```
def impotar_csv(nome):
    return pd.read_csv(nome)
dados = impotar_excel('titulacao_potenciometrica.xlsx')
```

2.1 Editando DataFrame -Index e renomear colunas

A edição de um dataframe é composto por dezenas de comandos da biblioteca do Pandas, infelizmente não será possível abordar devidamente todos eles, por isso acesse o link abaixo para obter alguns resumos desta abrangente biblioteca.

Acesso aos resumos de algumas bibliotecas do Python
https://drive.google.com/drive/folders/12Bjumxn3Cwf1Ykt7-cPKSDsKbf71NKN2?usp=sharing

2.1.1 Tamanho do dataframe

Vamos entender o tamanho de um dataframe através do comando **shape**.

[6]:
```
dados.shape[1]
```

[6]: 2

2.1.2 Métodos index e columns

Como acesar os nomes de colunas e estrutura do index.

[7]:
```
dados.columns[1]
```

[7]: 'pH'

2.1.3 Método reindex()

Reconstrói o index do dataframe - O comando df.index retorna uma *series* do índice do dataframe

[8]:
```
def gerar_index(df):
    return df.reindex(list(range(1,df.shape[0])))
dados = gerar_index(dados)
```

```
dados.head()
```

[8]:
```
   V(mL)    pH
1    1.0  3.65
2    2.0  3.93
3    3.0  4.12
4    4.0  4.32
5    5.0  4.47
```

2.1.4 Renomear colunas

df.rename(columns={'nome_velho':'nome_novo'})

[9]:
```
def renomear_colunas(df):
    return df.rename(columns={df.columns[0]:'V', df.columns[1]:'pH'})
dados = renomear_colunas(dados)
dados.head()
```

[9]:
```
     V    pH
1  1.0  3.65
2  2.0  3.93
3  3.0  4.12
4  4.0  4.32
5  5.0  4.47
```

2.2 Obtendo informações de um dataframe.

Esta biblioteca fornece métodos que entregam informações de **tendência central e de dispersão** que são importantes para tomar algumas decisões com os dados. Além disso, a biblioteca inclui algumas opções de **visualização de dados**, como, gráficos tipo *scatter*, *box-plot*, *barras* e *histograma*.

2.2.1 Análise descritiva dos valores de pH

Vamos verificar os dados que são oferecidos com o método de descrição

2.2.2 df.describe()

[12]:
```
dados.describe()
```

[12]:
```
                V         pH
count  36.000000  36.000000
mean   18.125000   7.850556
std    10.127315   3.304809
min     1.000000   3.650000
25%     9.750000   5.025000
50%    18.500000   6.410000
75%    26.250000  11.497500
```

```
max    35.000000  12.790000
```

[11]:
```
dados['pH'].std()
```

[11]:
```
3.3048094342677907
```

2.2.3 Adicionando coluna com desvio padrão de pH ao DataFrame

[17]:
```
dados['Desvio Padrão'] = dados['pH'].std()
dados.head()
```

[17]:
```
     V    pH  Desvio Padrão
1  1.0  3.65       3.304809
2  2.0  3.93       3.304809
3  3.0  4.12       3.304809
4  4.0  4.32       3.304809
5  5.0  4.47       3.304809
```

[19]:
```
dados['std'] = dados['Desvio Padrão'].round(2)
dados.head()
```

[19]:
```
     V    pH  Desvio Padrão  std
1  1.0  3.65       3.304809  3.3
2  2.0  3.93       3.304809  3.3
3  3.0  4.12       3.304809  3.3
4  4.0  4.32       3.304809  3.3
5  5.0  4.47       3.304809  3.3
```

2.2.4 Excluindo uma coluna através do método df.drop()

df.drop("Nome_da_coluna", axis = 'columns')

Lembrando que:

axis=0 –> index
axis=1 –> columns

[20]:
```
dados = dados.drop('Desvio Padrão', axis=1)
dados.head()
```

[20]:
```
     V    pH  std
1  1.0  3.65  3.3
2  2.0  3.93  3.3
3  3.0  4.12  3.3
4  4.0  4.32  3.3
5  5.0  4.47  3.3
```

Para excluir linhas de dataframe, utilize index para 1 linha ou o slice para um intevalo de linhas. Observe abaixo:

[22]:
```
dados.drop(dados.index[0:3],axis=0).head()
```

[22]:

	V	pH	std
4	4.0	4.32	3.3
5	5.0	4.47	3.3
6	6.0	4.60	3.3
7	7.0	4.71	3.3
8	8.0	4.85	3.3

2.3 Visualizando dados com pandas

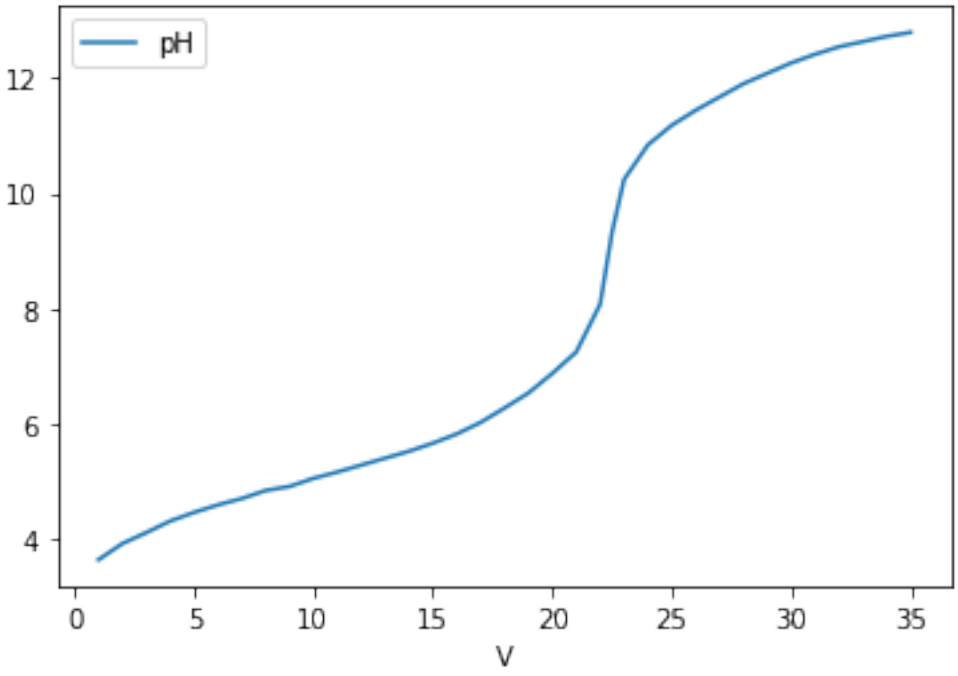
Embora os dados experimentais não são adequados para os tipos de representações gráficas abaixo, aplicaremos suas representações gráficas.

2.3.1 Gráfco de disperção *Scatter*

`df.plot.scatter(x,y)`

[330]:
```
dados.plot(x='V',y='pH')
```

[330]: <AxesSubplot:xlabel='V'>

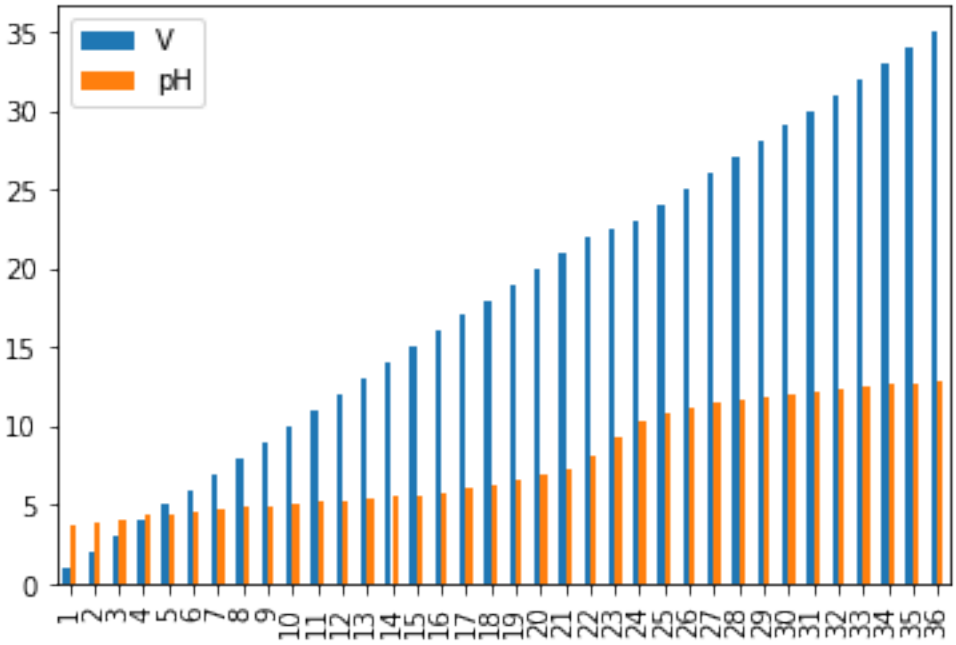


2.3.2 Gráfco de Barras

`df.plot.bar()`

[331]:
```
dados.plot.bar()
```

[331]: <AxesSubplot:>

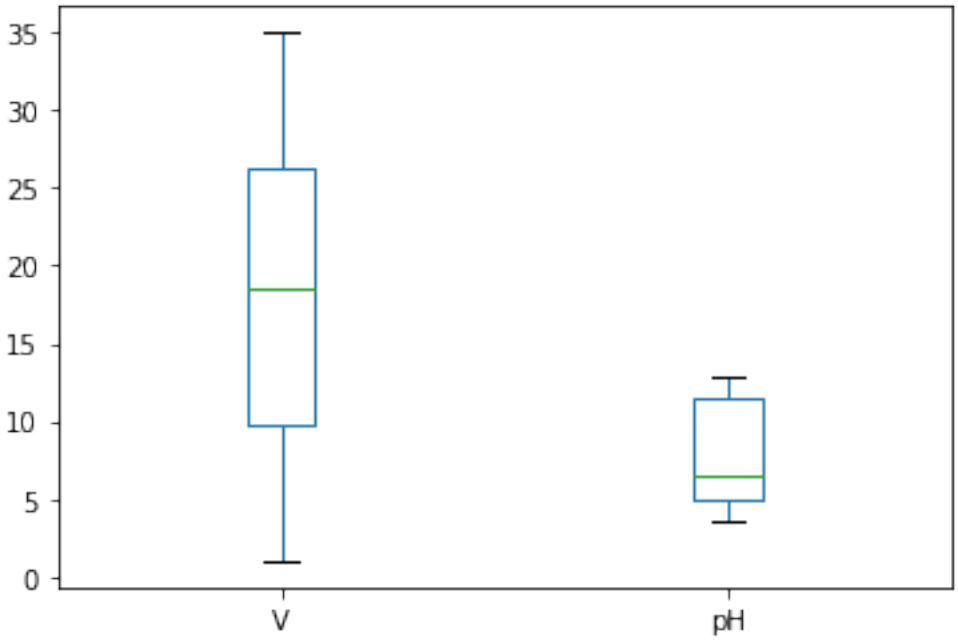


2.3.3 Boxplot

`df.plot.box()`

[332]:
```
dados.plot.box()
```

[332]: <AxesSubplot:>

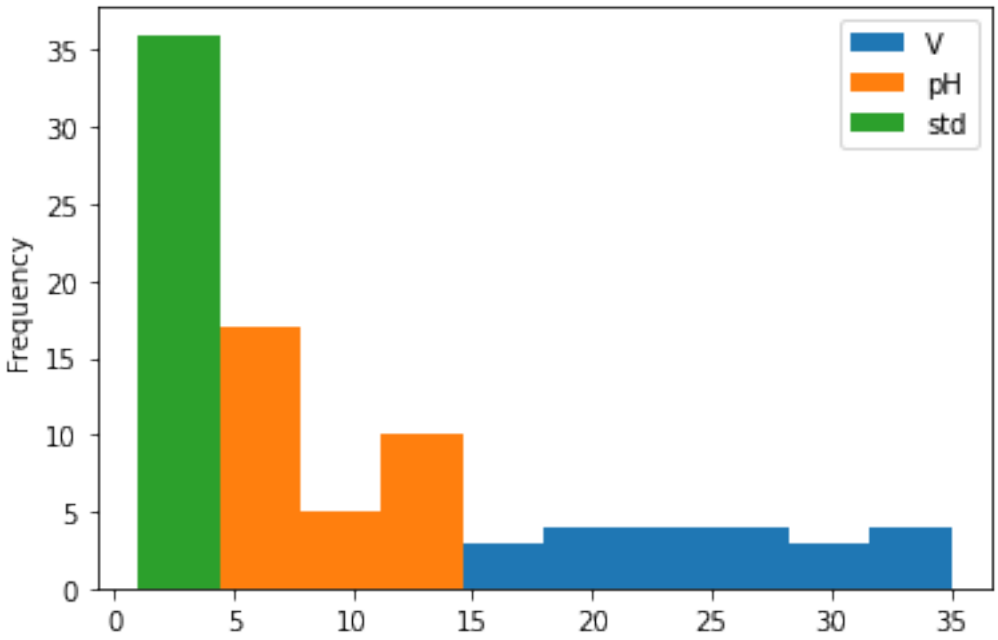


2.3.4 Histograma

`df.plot.hist()`

[24]:
```
dados.plot.hist()
```

[24]: `<AxesSubplot:ylabel='Frequency'>`



2.3.5 Aviso contextual:

Vale salientar que as representações gráficas apresentadas não possuem algum valor na aplicação de titulação potenciométrica, estamos aproveitando este exemplo para apresentar alguns recursos oferecidos pela biblioteca Pandas.

3 2º Passo: Tratamento dos dados

Nesta etapa **trataremos os dados** para obter as colunas com os valores da **primeira derivada**, **primeira derivada reversa** e **segunda derivada**.

3.1 Gerando colunas com os dados das derivadas

Nesta etapa precisaremos explorar novos métodos da biblioteca Pandas para realizar o **cálculo das derivadas**, neste caso o problema está relacionado em aplicar estes cálculos diretamente aos dados experimentais, ou seja, sem utilizar uma equação que descreva o gráfico. Assim, utilizaremos os seguintes métodos:

`assign()`
Retorna o dataframe juntamente com uma nova coluna que foi gerado através de um kwarg.
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.assign.html

`eval()`
Avalia uma expressão matemática através de uma *string*, dessa maneira as operações matemáticas são suportadas dentro de uma string.
https://pandas.pydata.org/docs/reference/api/pandas.eval.html

`diff()`
Retorna o cálculo da derivada de dados selecionados.
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.diff.html?highlight=diff#pandas.DataFrame.di

[334]:
```
# Coluna primeira derivada
def derivar_d1(df):
    return df.assign(d1=df.diff().eval('pH/V').round(2))
dados = derivar_d1(dados)
```

[335]:
```
# Coluna primeira derivada reversa
def derivar_d1r(df):
    return df.assign(d1r=df.diff().eval('V/pH').round(2))
dados = derivar_d1r(dados)
```

[336]:
```
# Coluna segunda derivada
def derivar_d2(df):
    return df.assign(d2=df.diff().eval('d1/V').round(2))
dados = derivar_d2(dados)
```

```
dados.head()
```

[336]:

	V	pH	d1	d1r	d2
1	1.0	3.65	NaN	NaN	NaN
2	2.0	3.93	0.28	3.57	NaN
3	3.0	4.12	0.19	5.26	-0.09
4	4.0	4.32	0.20	5.00	0.01
5	5.0	4.47	0.15	6.67	-0.05

4 3º Passo: Visualização dos dados

Precisamos criar uma função que retone em uma única imagem todos os gráficos relacionados à **Titulação Potenciométrica**, para isso, vamos abordar o recurso **pyplot** de **plt.subplot()**. Esta ferramenta possibilita **várias combinações para representação multigráfica**, esta função vai ser muito importante quando desejarmos criar um recurso para salvar os dados, por exemplo, armazenar os dados em um arquivo pdf.

Material Complementar:

Matplotlib.Pyplot
https://matplotlib.org/stable/gallery/lines_bars_and_markers/categorical_variables.html#sphx-glr-gallery-lines-bars-and-markers-categorical-variables-py

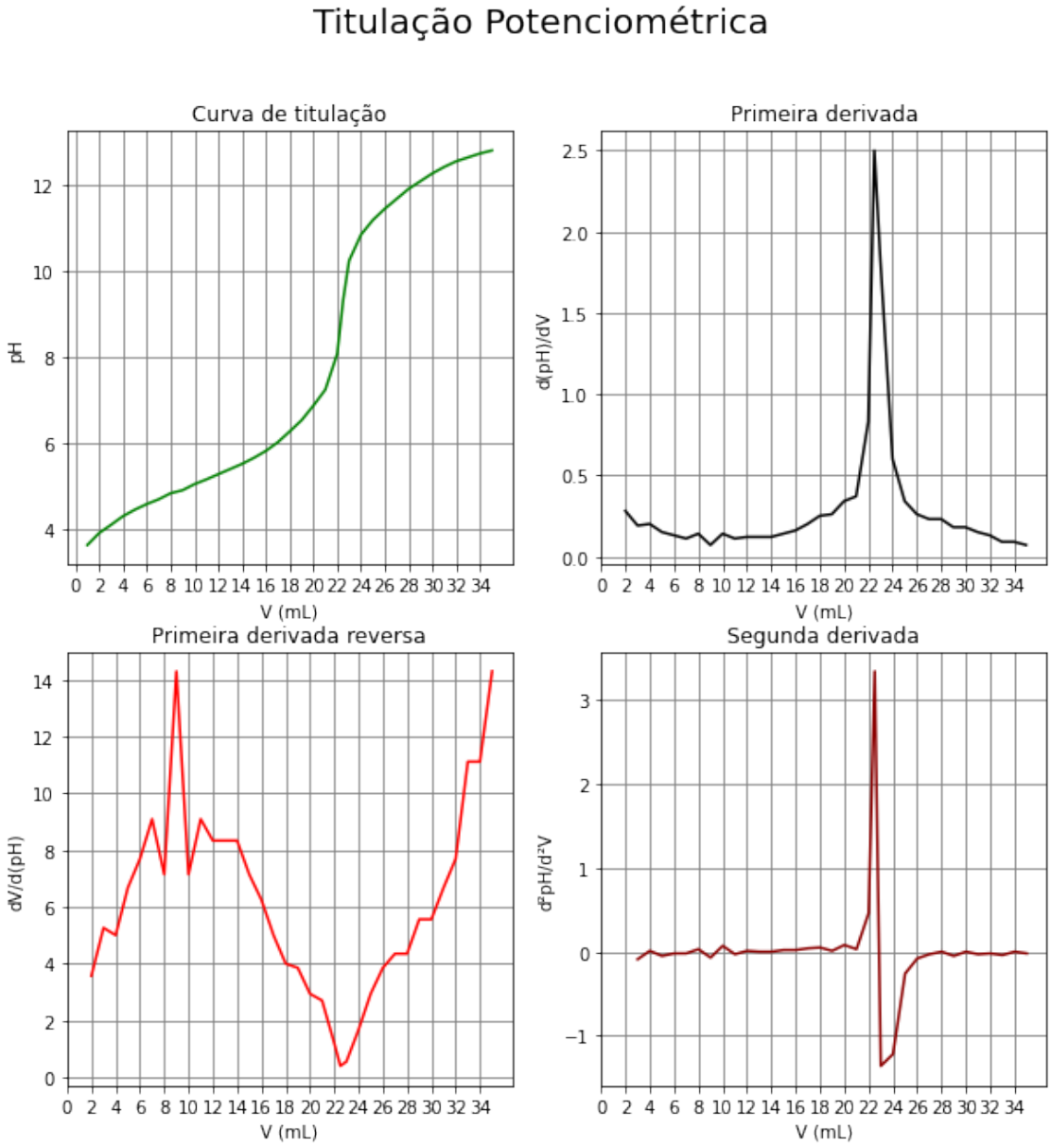
Referência Style Sheets
https://matplotlib.org/3.5.1/gallery/style_sheets/style_sheets_reference.html

[337]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[338]:
```
def plotar_grafico(df):
    fig, axs = plt.subplots(2,2, figsize=(10,10))

    axs[0,0].plot(df['V'],df['pH'], color='green' )
    axs[0,1].plot(df['V'],df['d1'],color='k')
    axs[1,0].plot(df['V'],df['d1r'], color= 'r')
    axs[1,1].plot(df['V'],df['d2'], color= 'darkred')

    axs[0,0].grid(color='gray', linestyle='solid')
    axs[0,1].grid(color='gray', linestyle='solid')
    axs[1,0].grid(color='gray', linestyle='solid')
    axs[1,1].grid(color='gray', linestyle='solid')

    axs[0,0].set_title('Curva de titulação')
    axs[0,1].set_title('Primeira derivada')
    axs[1,0].set_title('Primeira derivada reversa')
    axs[1,1].set_title('Segunda derivada')
```

```
    axs[0,0].set_xticks(np.arange(0,35,step=2))
    axs[0,1].set_xticks(np.arange(0,35,step=2))
    axs[1,0].set_xticks(np.arange(0,35,step=2))
    axs[1,1].set_xticks(np.arange(0,35,step=2))

    axs[0,0].set_xlabel('V (mL)')
    axs[0,1].set_xlabel('V (mL)')
    axs[1,0].set_xlabel('V (mL)')
    axs[1,1].set_xlabel('V (mL)')

    axs[0,0].set_ylabel('pH')
    axs[0,1].set_ylabel('d(pH)/dV')
    axs[1,0].set_ylabel('dV/d(pH)')
    axs[1,1].set_ylabel('d²pH/d²V')

    fig.suptitle('Titulação Potenciométrica', fontsize=20)
    plt.savefig('titulacao_potenciometrica.pdf')

    return plt.show()
```

[339]:
```
plotar_grafico(dados)
```



5 4º Passo: Obtendo dados

Uma vez tratado e visualizado os dados, vamos obter os dados que estamos procurando. Em outras palavras, vamos obter o volume de equivalência através dos gráficos de derivadas.

5.1 Resolvendo os valores inválidos (NaN) no dataframe

O valor **NaN** significa que é um **valor inválido** em nossos dados, este tipo de valor se trata de um grande problema envolvendo ciência de dados porque muitas vezes os dados coletado possuem estes vazios. Para resolver isto, recomenda-se na **maioria** dos casos que **substitua por 0**. Aqui

na biblioteca pandas podemos resolver isto com o método **fillna()** ou **replace()**.

No entanto, substituir o valor inválido no **segundo gráfico** não é uma boa ideia, uma vez que o valor que estamos procurando para encontrar o volume de equivalência é para y=0. Logo, temos que substituir por outro valor que **não seja igual à zero**.

Material complementar da página GeekforGeeks:
https://www.geeksforgeeks.org/replace-nan-values-with-zeros-in-pandas-dataframe/

Material complementar da página Insight
https://insightlab.ufc.br/6-truques-do-pandas-para-impulsionar-sua-analise-de-dados

[340]:
```
def trocar_nan(df):
    df['d1'] = df['d1'].fillna(axis=0, method='bfill')  #ffill substituirá␣
↪valores anteriores enquando o bfill será valores posteriores
    df['d1r'] = df['d1r'].fillna(axis=0, method='bfill')
    df['d2'] = df['d2'].fillna(axis=0, method='bfill')
    return df
trocar_nan(dados).head()
```

[340]:

	V	pH	d1	d1r	d2
1	1.0	3.65	0.28	3.57	-0.09
2	2.0	3.93	0.28	3.57	-0.09
3	3.0	4.12	0.19	5.26	-0.09
4	4.0	4.32	0.20	5.00	0.01
5	5.0	4.47	0.15	6.67	-0.05

5.2 Obtendo volume de equivalência

Sabemos que o **volume de equivalência** é obtido pelos gráficos de primeira derivada, primeira derivada inversa e segunda derivada, o primeiro gráfico verificamos o **valor máximo global**, já o segundo quando o **valor mínimo global** e o terceiro quando **y=0**.

5.2.1 Manipulação de dados em um dataframe

Para encontrarmos o **volume de equivalência**, precisa-se encontrar o valor máximo da coluna da primeira derivadas para depois associar o volume correspondente. Este tipo de problema onde se procura um valor ou intervalor de um dataframe é muito comum, o método que utilizaremos será o método **df.loc[]**. Este método possibilita **encontrar a linha de um determinado valor** ou as **linhas de um intervalo**. Neste caso, o valor que precisamos é o valor máximo da coluna 'd1'.

Outro recurso que utilzaremos também nesta etapa o **df.values**, este retorna o *numpy.array* de um DataFrame ou Serie. Com isso, vamos acessar **o volume de equivalência** a através do **index** deste array, semelhante que fizemos com as listas.

[341]:
```
dados.loc[dados['d1'] == 0.12]
```

[341]:

	V	pH	d1	d1r	d2
12	12.0	5.29	0.12	8.33	0.01
13	13.0	5.41	0.12	8.33	0.00
14	14.0	5.53	0.12	8.33	0.00

5.2.2 Encontrando Volume de Equivalência no gráfico primeira derivada

[342]:
```
def ve_d1(df):
    return df.loc[df['d1'] == df['d1'].max()].values[0][0]
ve_d1(dados)

def pe_d1(df):
    return df.loc[df['d1'] == df['d1'].max()].values[0][1]
pe_d1(dados)
```

[342]: 9.33

5.2.3 Encontrando Volume de Equivalência no gráfico primeira derivada invertida

[343]:
```
def ve_d1r(df):
    return df.loc[df['d1r'] == df['d1r'].min()].values[0][0]
ve_d1r(dados)
```

[343]: 22.5

5.2.4 Encontrando Volume de Equivalência no gráfico segunda derivada

[344]:
```
def ve_d2(df):
    return df.loc[df['d2'] == 0].values[0][0]
ve_d2(dados)
# valor incorreto
```

[344]: 13.0

5.3 Explorando o problema encontrado no gráfico da segunda derivada.

Notamos que o gráfico da segunda derivada, por algum motivo, possui mais de um valor nulo no seu eixo das ordenadas. Com isso, precisamos selecionar a região de interesse para obter um resultado plausível.

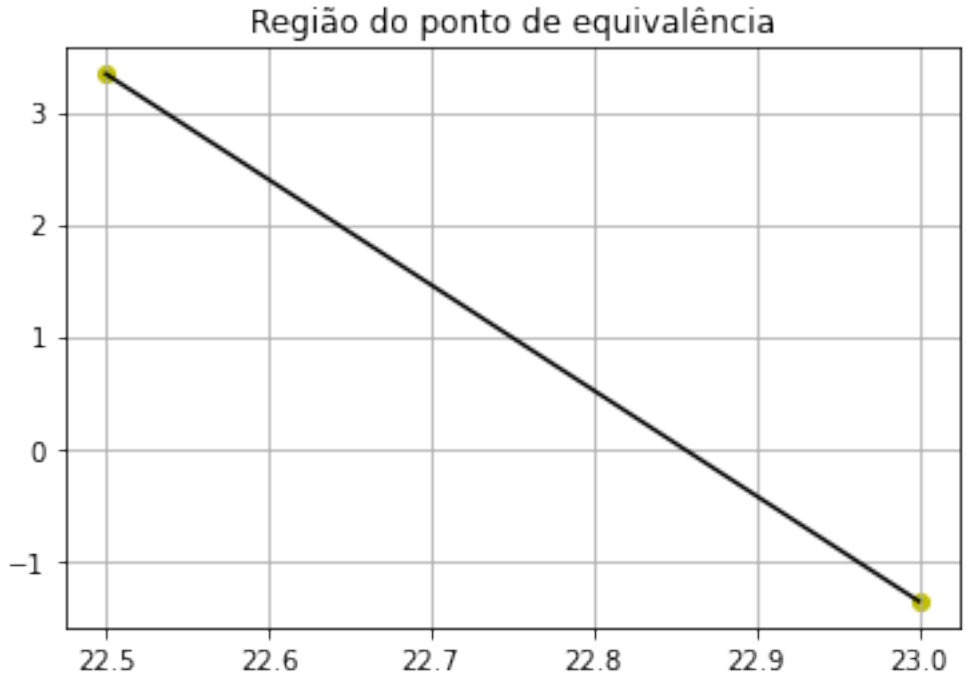
5.3.1 Selecionando um intervalo de dados

Vamos selecionar um intervalo que inclua a zona de máximo/mínimos e que seja próximo ao ponto de equivalência encontrado no gráfico de derivadas.

[345]:
```
Vmax = dados.loc[dados['d2'] == dados['d2'].max()].index[0]
Vmin = dados.loc[dados['d2'] == dados['d2'].min()].index[0]
```

```
x_r = dados.loc[Vmax:Vmin]['V'].values
y_r = dados.loc[Vmax:Vmin]['d2'].values
```

[346]:
```
plt.scatter(x_r,y_r, color ='y')
plt.plot(x_r, y_r, color = 'k')
plt.title('Região do ponto de equivalência')
plt.grid()
plt.show()
```



5.3.2 Interpolação de gráfico 1-D

Agora visualizamos o gráfico da segunda derivada na região de interesse, precisamos obter o ponto deste gráfico onde **y=0**. Então, precisamos criar uma **interpolação** para gerar **n pontos** neste gráfico, tendo em vista isso, vamos recorrer o módulo **scipy.interpolate**.

Interpolação com módulo Scipy da página GeekforGeeks
https://www.geeksforgeeks.org/scipy-interpolation/

Sobre interpolação**:**
https://en.wikipedia.org/wiki/Linear_interpolation

Documentação do módulo scipy.interpolate**
https://docs.scipy.org/doc/scipy/reference/interpolate.html

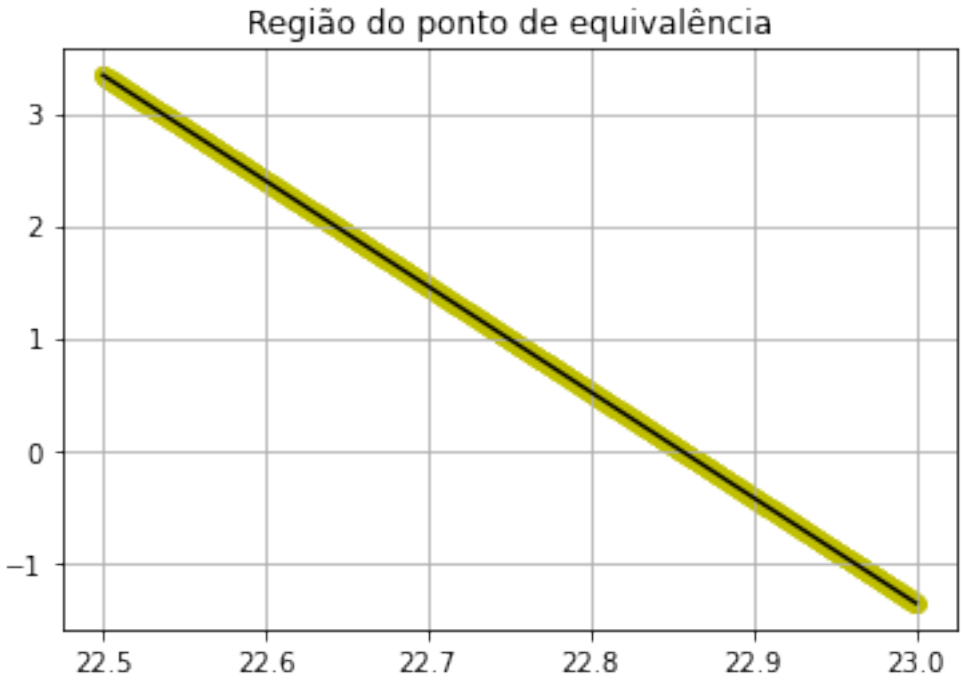
[347]:
```
from scipy.interpolate import interp1d
import numpy as np
```

[348]:
```
x = dados['V'].values
y = dados['d2'].values
```

[349]:
```
f = interp1d(x,y)
```

[350]:
```
x_i =  np.linspace(22.5,23, num = 1000, endpoint=True)
y_i = f(x_i)
interp = pd.DataFrame({'V':x_i.round(2),'d2':y_i.round(2)})
```

[351]:
```
plt.scatter(x_i,y_i, color ='y')
plt.plot(x_r, y_r, color = 'k')
plt.title('Região do ponto de equivalência')
plt.grid()
plt.show()
```



[352]:
```
ve_d2(interp)
```

[352]: 22.85

5.3.3 Regressão Linear

Através do método de interpolação 1d utilizado na biblioteca **scipy**, foi possível encontrar o volume de equivalência. Vamos continuar explorando este erro para apresentá-los a **regressão linear** usando a biblioteca **sklearn**(). Vale salientar, que esta ferramenta não é necessária para prosseguir com esta aplicação química, no entanto, é muito comum precisarmos construir uma equação de uma reta em diversas aplicações.

Documentação Sklearn:
https://en.wikipedia.org/wiki/Linear_regression

Sobre regressão linear:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Regressão Linear com Sklearn: Conceito e Aplicação (Medium):
https://medium.com/@lamartine_sl/regress%C3%A3o-linear-com-sklearn-modelo-de-previs%C3%A3o-de-custos-com-plano-de-sa%C3%BAde-5e963e590f4c)

[353]:
```
from sklearn.linear_model import LinearRegression
```

[354]:
```
reg = LinearRegression()
```

[355]:
```
x_i = x_i.reshape(-1,1)
```

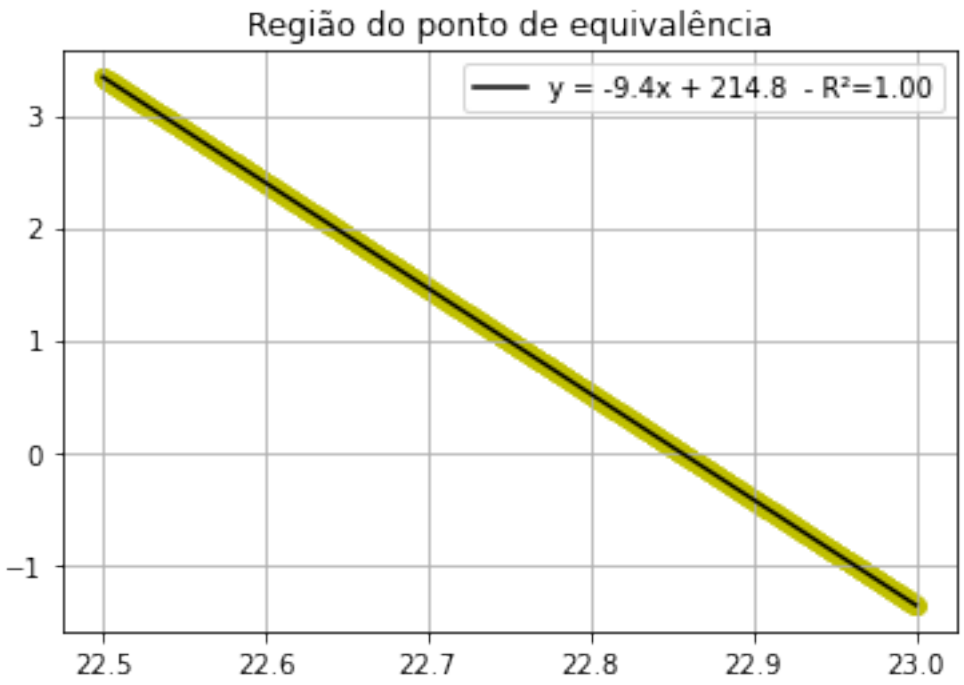
[356]:
```
reg = reg.fit(x_i,y_i)
```

[357]:
```
coef_a = reg.coef_[0]
```

[358]:
```
coef_l = reg.intercept_
```

[359]:
```
coef_d = reg.score(x_i,y_i)
```

[360]:
```
plt.scatter(x_i,y_i, color ='y')
plt.plot(x_r, y_r, color = 'k')
plt.title('Região do ponto de equivalência')
plt.grid()
plt.legend(['y = {0:.1f}x + {1:.1f}  - R²={2:.2f}'.format(coef_a,␣
 ↪coef_l,coef_d)])
plt.show()
```



[363]:
```
v_e = round(-coef_l/coef_a,2)
v_e
```

[363]: 22.86

6 Último Passo: Apresentando Resultados

Agora temos todos os resultados que o experimentador deve saber para tirar suas conclusões de sua análise. Entretanto, vamos criar um módulo contendo as definições desta aplicação, isto significa que é importante que este módulo entregue os resultado de forma adequada. Assim sendo, vamos entregar as **imagens dos gráficos** em uma extensão de preferência e os **dados das derivadas e volume de equivalência** em uma **planilha Excel**.

[375]:
```
def gerar_df_ve(df):
    return pd.DataFrame({'Primeira Derivada': [ve_d1(df)],'Primeira Derivada.␣
 ↪R':[ve_d1r(df)],
                         'Segunda Derivada':[ve_d2(df)]},index=['V(mL)'])
gerar_df_ve(dados)
```

[375]:

	Primeira Derivada	Primeira Derivada. R	Segunda Derivada
V(mL)	22.5	22.5	13.0

[376]:
```
def salvar_df(df):
    resultados = pd.ExcelWriter('titulacao_potenciometrica.xlsx')
```

```
    df.to_excel(resultados, sheet_name='Titulação Dados')
    gerar_df_ve(df).to_excel(resultados, sheet_name='Volume de Equivalência')
    return resultados.save()
```

[377]:
```
salvar_df(dados)
```