1. Escreva uma função que receba uma lista de números e retorne outra lista com os números ímpares.

Sabemos que todo número ímpar m ∈ Z pode ser escrito na forma m=2K+1, onde K ∈ Z, e o resto da divisão de m por 2 é diferente de zero. Logo, para o nosso código, vamos fazer duas verificações com o while True:

- se há números repetidos na lista inserida pelo usuário
- se o resto da divisão dos números é zero

Feitas as verificações, basta armazenar os novos valores em uma nova lista.

```
while True:
    try:
        tam=int(input("Quantos valores terão na lista?:  "))
        if tam<0:
            print("O número deve ser inteiro +. Digite novamente:
")
            continue
        break
    except ValueError:
        print("Digite um caractere válido: números inteiros +:  ")
        continue
intervalo=[]
for i in range(1,tam+1):
        while True:
            try:
                num=int(input("Digite um número:  "))
                if num in intervalo:
                    print("Número repetido! Digite outro:  ")
                    continue
                else:
                    intervalo.append(num)
                break
            except ValueError:
                    print("Digite caracteres válidos (inteiros + ou
-)")
                continue
def selec_impares(intervalo):
    return[num for num in intervalo if num%2!=0]
impares=selec_impares(intervalo)
print("Os números ímpares da sua lista original são: ",impares)
```

2. Escreva uma função que receba uma lista de números e retorne outra lista com os números primos presentes.

Sabemos que todo número primo só é divisível por 1 e por ele mesmo. Para fazer uma verificação otimizada, precisamos definir até onde será estendido o laço de repetição. Como todo número inteiro pode ser escrito da forma n=x*y, onde necessariamente um é maior ou igual ao outro. Portanto, no caso onde x=y, $n=x^2$

⇔ x=sqrt(n). Então, podemos definir o range do laço de repetição como a raíz quadrada do número inserido “n”. Se o resto da divisão for zero, então o número não é primo.

```
primeiro=int(input("Insira o primeiro número:  "))
segundo=int(input("Insira o segundo número:  "))

nr_lista=list(range(primeiro, segundo+1))
nr_par=[number for number in nr_lista if number%2==0]
nr_impar=[number for number in nr_lista if number%2!=0]

nr_nao_primo=[]
for n in nr_lista:
    if n>=2:
        for i in range(2,int(n**0.5)+1):
            if n%i==0:
                nr_nao_primo.append(n)
                break
nr_primos=[number for number in nr_lista if number>1 and number
not in nr_nao_primo]

#print(f"\nA lista original inserida é: {nr_lista}")
print(f"\nA lista de números primos é: {nr_primos}")
```

3. Escreva uma função que receba duas listas e retorne outra lista com os elementos que estão presentes em apenas uma das listas.

Para o nosso código, vamos fazer duas verificações (para além da validade do valor inserido pelo usuário - while True -):

- se as listas são iguais
- se existem elementos únicos em alguma das listsa

```
while True:
    try:
        x=int(input("Quantos valores terão na lista?:  "))
        if x<0:
            print("O número digitado deve ser um inteiro maior que
zero. Digite novamente:  ")
            continue
        break
    except ValueError:
        print("Digite um caractere válido: números inteiros +:  ")
        continue
while True:
    try:
        y=int(input("Quantos valores terão na lista?:  "))
        if y<0:
            print("O n° deve ser um inteiro maior que zero. Digite
novamente:  ")
            continue
        break
    except ValueError:
        print("Digite um caractere válido: Números inteiros
positivos:  ")
```

```
        continue

lista_x=[]
lista_y=[]

for i in range(1,x+1):
        while True:
            try:
                num_x=float(input(f"Digite {x} números para
preencher a primeira lista:  \n"))
                lista_x.append(num_x)
                break

            except ValueError:
                print("Digite caracteres válidos (inteiros+ ou
-)")
                continue
for i in range(1,y+1):
        while True:
            try:
                num_y=float(input(f"Digite {y} números para
preencher segunda lista:  \n"))
                lista_y.append(num_y)
                break

            except ValueError:
                print("Digite caracteres válidos (inteiros + ou
-)")
                continue

def iguais(lista_x,lista_y):
     return [elemento for elemento in lista_x if elemento in
lista_y]

lista_iguais=iguais(lista_x,lista_y)

print(f"\nPrimeira lista: {lista_x}\nSegunda lista: {lista_y}\n-->
Números iguais entre lista x e lista y: {lista_iguais}")
```

4. Dada uma lista de números inteiros, escreva uma função para encontrar o segundo maior valor na lista.

Para o nosso código, precisamos verificar se o tamanho das listas é válido (>=2) e se os valores inseridos também são válidos (números inteiros). Para isso, vamos fazer as verificações com o While True:

```
def segundo_maior(lista):
    if len(lista)<2:
        return None
    lista_unica =list(set(lista))
    if len(lista_unica)<2:
        return None

    lista_ordenada=sorted(lista_unica, reverse=True)
    return lista_ordenada[1]

def obter_lista():
```

```
    while True:
        try:
            tamanho=int(input("Quantos n° você quer inserir?:  "))
            if tamanho>=2:
                break
            else:
                print("Número inválido!\nA lista deve ter pelo
menos 2 números.\n")
        except ValueError:
            print("Entrada inválida. Digite um número inteiro.")

    numeros=[]
    for i in range(tamanho):
        while True:
            try:
                num=int(input(f"Digite o número {i+1}: "))
                numeros.append(num)
                break
            except ValueError:
                print("Valor inválido. Digite um número inteiro:
")
    return numeros

lista = obter_lista()
resultado = segundo_maior(lista)

if resultado is not None:
    print(f"\nO segundo maior valor é: {resultado}")
else:
    print("\nNão há segundo maior valor válido na lista.")
```

5. Crie uma função que receba uma lista de tuplas, cada uma contendo o nome e a idade de uma pessoa, e retorne a lista ordenada pelo nome das pessoas em ordem alfabética.

Para o nosso código, vamos fazer duas verificações com o While True: se a quantidade de pessoas inserida para preencher as listas é um número inteiro e se a idade inserida também é um número inteiro (e positivo).

```
def lista_pessoas():
    lista=[]
    while True:
        try:
            n=int(input("Quantas pessoas vão ser adicionadas? "))
            if n>0:
                break
            else:
                print("ERRO. Por favor, insira um número inteiro
positivo.")
        except ValueError:
            print("ERRO. Digite um número inteiro.")
    for i in range(n):
        nome=input(f"Digite o nome da pessoa {i+1}: ")
        while True:
            try:
```

```
                idade=int(input(f"Digite a idade da pessoa{i+1}:
"))
                break
            except ValueError:
                print("ERRO. Digite uma idade válida (número
inteiro maior que zero).")

        lista.append((nome, idade))
    return lista

def ordenar(lista):
    return sorted(lista, key=lambda pessoa: pessoa[0].lower())

pessoas=lista_pessoas()
ordenadas=ordenar(pessoas)

print("\nLista ordenada por nome:")
for nome, idade in ordenadas:
    print(f"{nome} - {idade} anos")
```

6. Como identificar e tratar outliers em uma coluna numérica usando desvio padrão ou quartis?

Se você tiver o desvio padrão e a média dos dados em determinada coluna, você pode utilizar o método Z-Score para descobrir os outliers. Por sua vez, de posse dos quartis, você pode utilizar o método de Tukey para encontrar os outliers. Após a sua delimitação, o tratamento dos valores atípicos vai depender do tipo de análise que está sendo feita: o mais comum é substituir esses valores pela mediana, pela média ou ainda, embora menos comum, por zero.

Entretanto, em se tratando de análises categóricas, **talvez seja interessante criar uma nova categoria para o outlier**. Por exemplo: análise da distribuição espacial de renda, segundo dados dos setores censitários do IBGE, em determinado bairro de Fortaleza-CE. Em um cenário onde a média salarial é X e um pequeno grupo ganha mais do que 3X, é interessante ver como a distribuição espacial acontece sem os outliers para então inserir uma categoria para estes. Assim, é possível reconhecer visualmente onde se concentram as distribuições anômalas.

7. Como concatenar vários DataFrames (empilhando linhas ou colunas), mesmo que tenham colunas diferentes? Dica: Utiliza-se pd.concat() especificando axis=0 (linhas) ou axis=1 (colunas). Quando há colunas diferentes, os valores ausentes são preenchidos com NaN.

```
import pandas as pd
df_a=pd.read_csv("Nome_do_arquivo_a.csv")
df_b=pd.read_csv("Nome_do_arquivo_b.csv")

df_concat_ab_linhas=pd.concat([df_a,df_b],axis=0,ignore_index=True
)
#para concatenar por linhas

ou:
```

```
df_concat_ab_colunas=pd.concat([df_a,df_b],axis=1,ignore_index=Tru
e)
#para concatenar por colunas
```

8. Utilizando pandas, como realizar a leitura de um arquivo CSV em um DataFrame e exibir as primeiras linhas?

```
import pandas as pd
df=pd.read_csv("Nome_do_arquivo.csv")
print(df.head()) #imprime o resultado das 5 primeiras linhas na
tela. Caso deseje ver mais linhas, colocar o número dentro do
parêntese. ex.: df.head(12)
```

9. Utilizando pandas, como selecionar uma coluna específica e filtrar linhas em um "DataFrame" com base em uma condição?

```
import pandas as pd
df=pd.read_excel("Nome_do_arquivo.xlsx")
df["Nome da Coluna"] #selecionando uma coluna específica

df_filtrado=df[df["Nome da Coluna"]== "condicao"] #filtrando a
coluna com base em uma condição qualquer

print(df_filtrado.head()) #imprime o resultado de todas as colunas
e das cinco primeiras linhas na tela. Recomenda-se fazer isso para
poupar processamento em casos de dataframes muito grandes.

#Caso queira atualizar o dataframe inicial, escrever
df=df[df["Nome da Coluna"]== "condicao"] e print(df).
```

10. Utilizando pandas, como lidar com valores ausentes (NaN) em um DataFrame?

Supondo que o seu dataframe tenha sido carregado no arquivo como `"df"`, basta utilizar o `df.fillna(0)` para que o NaN seja substituído por zero ou por algum outro valor ou condição no seu código `(df.fillna(outro_valor_ou_condicao))`.

```
df_atualizado=df.fillna(0) #caso você queira armazenar essa
atualização de valores em um novo dataframe

df=df.fillna(0) #caso você queira atualizar os valores dentro do
próprio dataframe
```