Faculdade IMPACTA	**Programação Orientada a Objetos** **Tratamento de Exceções e Testes de Unidade** **Exercícios**

Exercício 01

Nesta questão você deve identificar as partes problemáticas do código e reescrevê-lo utilizando tratamento de exceções.
Ou seja, devem ser identificadas todas as exceções que podem ser geradas e, para cada uma, deve ser dado o tratamento adequado que, nesse exercício, significa alertar o usuário quanto ao problema.

```
x = int(input('Primeiro valor:'))
y = int(input('Segundo valor:'))
z = x / y
print('O resultado da divisão é:', z)
```

Exercício 02

O código abaixo lança uma exceção e interrompe a execução do programa.
Utilizando tratamento de exceções, corrija o programa com o objetivo de alertar o usuário sobre o erro ocorrido, e impedir que o programa seja interrompido bruscamente.

```
def funcao_1():
    print('Início da função')
    lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    for i in range(15):
        print(lista[i])
    print('Fim da função')

print('Início do programa')
funcao_1()
print('Fim do programa')
```

Exercício 03

Preencha uma lista com 5 nomes de pessoas, informados pelo usuário.

a) Criar uma função que recebe como parâmetro de entrada a lista e uma posição (índice) dessa lista e retorna o nome que está nessa posição.
   - Essa função deve gerar e tratar uma exceção do tipo **`IndexError`** caso o índice não exista na lista.

Exercício 04

Crie um dicionário para armazenar uma listagem de alunos.

a) Utilize como chave o RA do aluno e como valor o nome do aluno.
b) Os dados devem ser informados pelo usuário
c) O RA de cada aluno deve ser composto por um número inteiro de exatamente 7 dígitos.
   - Caso o RA informado não esteja no formato correto, deve ser gerada uma exceção do tipo **`ValueError`**
   - Caso o RA informado já exista no dicionário, deve ser gerada uma exceção do tipo **`TypeError`**

Observação: Você pode utilizar o código abaixo como exemplo e alterá-lo para gerar e tratar as exceções solicitadas.

```
alunos = {}
for i in range(5):
    ra = int(input('RA: '))
    nome = input('Nome: ')
    alunos[ra] = nome
print(alunos)
```

Exercício 05

Importe o módulo abaixo para um programa de teste e escreva testes unitários para as funções do módulo:

```
def converte_para_celsius(fahrenheit):
    celsius = (5.0/9.0) * (fahrenheit - 32)
    return celsius


def converte_para_fahrenheit(celsius):
    fahrenheit = 1.8 * celsius + 32
    return fahrenheit
```

Utilize os valores abaixo como parâmetros de entrada e saída das funções.

converte_para_fahrenheit(celsius)	
Entrada	***Saída***
0	32.0
27	80.6
95	203.0

converte_para_celsius(fahrenheit)	
Entrada	***Saída***
32	0
41	5.0
95	35.0

Exercício 06

Construa um módulo de nome **`numero_perfeito.`** Nesse módulo, defina uma função chamada **`eh_perfeito(n).`**
Essa função deve retornar **`True`** se ***n*** for um número perfeito e retornar **`False`** se ***n*** não for um número perfeito.
OBS: Um número perfeito é um número inteiro para o qual a soma dos seus divisores positivos (excluindo ele mesmo) é igual ao próprio número. Por exemplo:

O número 6 é um número perfeito, pois: 6 = 1 + 2 + 3.
O número 28 também é um número perfeito, pois: 28 = 1 + 2 + 4 + 7 + 14

Utilize o arquivo de teste a seguir para testar a sua implementação.

```
import numero_perfeito

def primeiro_teste():
    try:
        retorno = numero_perfeito.eh_perfeito(6)
        assert retorno is True
        print("Primeiro Teste Correto")
    except AssertionError:
        print("Primeiro Teste com Erro")

def segundo_teste():
    try:
        retorno = numero_perfeito.eh_perfeito(8128)
        assert retorno is True
        print("Segundo Teste Correto")
    except AssertionError:
        print("Segundo Teste com Erro")

def terceiro_teste():
    try:
        retorno = numero_perfeito.eh_perfeito(100)
        assert retorno is False
        print("Terceiro Teste Correto")
    except AssertionError:
        print("Terceiro Teste com Erro")

def quarto_teste():
    try:
        retorno = numero_perfeito.eh_perfeito(1)
        assert retorno is False
        print("Quarto Teste Correto")
    except AssertionError:
        print("Quarto Teste com Erro")
```

```
# O número zero não é perfeito
def quinto_teste():
    try:
        retorno = numero_perfeito.eh_perfeito(0)
        assert retorno is False
        print("Quinto Teste Correto")
    except AssertionError:
        print("Quinto Teste com Erro")

def sexto_teste():
    try:
        retorno = numero_perfeito.eh_perfeito(-6)
        assert retorno is False
        print("Sexto Teste Correto")
    except AssertionError:
        print("Sexto Teste com Erro")

primeiro_teste()
segundo_teste()
terceiro_teste()
quarto_teste()
quinto_teste()
sexto_teste()
```