Lógica de Programação Orientada a Objetos

Bem-vindos!

Objetivos

- EXCEÇÕES
- IMPORTAÇÃO

LET'S CODE

CAPTURANDO UMA EXCEÇÃO

```
try:
    execute_some_code()
except SomeException:
    handle_gracefully()
```

CAPTURANDO TODAS AS EXCEÇÕES

```
try:
    execute_some_code()
except:
    handle_gracefully()
```

DIVISÃO POR ZERO

- Tente dividir '1/0'. O que acontece?
- Capture a exceção e diga ao usuário que ele não pode dividir por zero.

CAPTURANDO MÚLTIPLAS EXCEÇÕES

- Lidando com todos eles do mesmo jeito

```
try:
    execute_some_code()
except (SomeException, AnotherException):
    handle_gracefully()
```

- Lidando com eles separadamente

```
try:
    execute_some_code()
except SomeException:
    handle_gracefully()
except AnotherException:
    do_another_thing()
```

LEVANTANDO EXCEÇÕES

- Exceções podem ser levantadas usando raise <exception> com argumentos opcionais.

```
raise RuntimeError()
raise RuntimeError("error message")
```

ACESSANDO UMA EXCEÇÃO

- Use as para acessar o objeto do tipo de exceção

```
try:
    raise RuntimeError("o hai")
except RuntimeError as e:
    print(e)
```

PROPAGANDO EXCEÇÕES

- Blocos try podem ser aninhados; Todas as exceções se propagam para o "manipulador de exceção raiz" de nível superior, se não são detectadas.
- O manipulador de exceção raiz (padrão) termina o processo Python.

```
try:
    try:
        raise Exception
    except Exception:
        print('Inner')
except Exception:
    print('Outer')
```

PROPAGANDO EXCEÇÕES

- Propagação pode ser forçado usando raise sem argumentos. Isso aumenta a exceção mais recente.
- Isso é útil para por exemplo, logging de exceção.

```
try:
    try:
        try:
            raise Exception
        except Exception:
            print('Inner')
            raise
except Exception:
    print('Outer')
```

FINALLY

- O código no bloco finally, deve sempre ser executado (a não ser que o Python quebre completamente).

```
try:
    open_file()
except IOError:
    print('Exception caught')
finally:
    close_file()
```

ELSE

- Código no bloco else deve ser executado quando nenhuma exceção foi levantada

```
try:
    open_file()
except IOError:
    print('Exception caught')
else:
    print('Everything went according to plan')
```

A DECLARAÇÃO IMPORT

- Habilita uso do outro arquivo python ou biblioteca
- Importação: import math
- Importação nomeada: import math as m
- Importação específica: from math import pow
- Importação total: from math import * (cuidado! use somente em casos específicos)

PACOTES

- Pacotes são namespaces que contém múltiplos pacotes e módulos.
- Pacotes são simplesmente diretórios, mas tem um porém: cada pacote/diretório DEVE conter um arquivo especial chamado de __init__.py
- Não inserindo o arquivo __init__.py em um pacote usando Python 3 deve funcionar mas isso é uma outra história

```
# fruits/__init__.py
# -- vazio -- nada aqui -- realmente nada -- somente um solitário e vazio arquivo

# fruits/apple.py
def print_it():
    print('apple')

# main.py
from fruits import apple
apple.print_it()
```

Se a pasta conter um arquivo __init__.py, ele mesmo pode ser importado com o nome do pacote.

```python
# foo/__init__.py
def greeting():
    return "Hello World!"

# main.py
from foo import greeting
print(greeting())
```

MÓDULOS SÃO SINGLETONS

```python
# stuff.py
fruits = ['Pineapple']

# module_a.py
import stuff
def foo():
    stuff.fruits.append('Apple')

# module_b.py
import stuff
def foo():
    stuff.fruits.append('Banana')
```

```
# program.py
import module_a, module_b, stuff
module_a.foo()
module_b.foo()
print(stuff.fruits)

# output
['Pineapple', 'Apple', 'Banana']
```

LET'S CODE

EXERCÍCIO

- https://www.hackerrank.com/challenges/exceptions/problem
- https://www.hackerrank.com/challenges/calendar-module/problem

LET'S CODE