Reading Python Programming

Módulos, Exceções e Manipulação de Arquivos

11 de julho de 2024

Agenda da Sessão

01. Módulos
02. unittest e doctest
03. Exceções
04. Manipulação de Arquivos

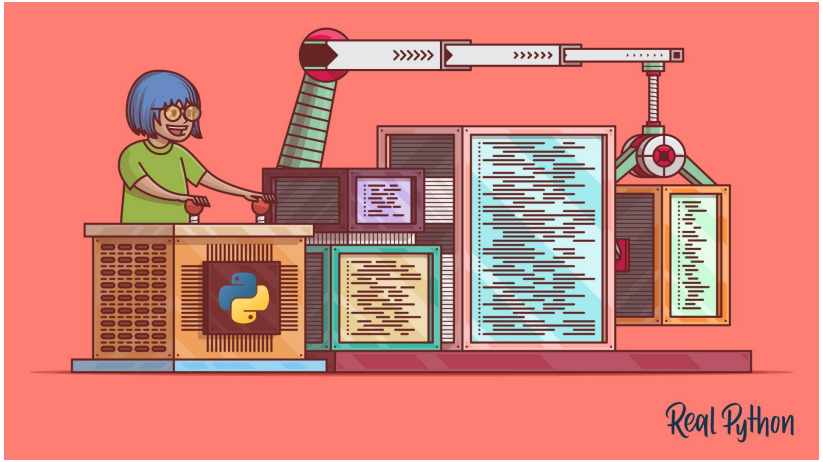
01 Módulos

Rumos

Módulos

Contextualização

- Os módulos contêm funções, classes e variáveis que podem ser importadas para outros locais do código
- Utiliza a keyword import que identifica o módulo e permite expandir as funcionalidades para o ficheiro atual
- Os módulos podem ser terceirizados ou criados pelo próprio programador, permitindo maior legibilidade e escalabilidade do código.



Módulos

Tipos de Importação e Sintaxe

Importar todo o módulo

- import {módulo}

Importar partes do módulo

- from {módulo} import {método}, {variável}

Alterar o nome utilizado no atual ficheiro

- import {módulo} as {novo_nome_módulo}

```
import math

from math import pi, ceil

import math as calc
```

Organização

- Ordenam-se por bibliotecas “oficiais”, bibliotecas terceirizadas e bibliotecas locais.

Módulos

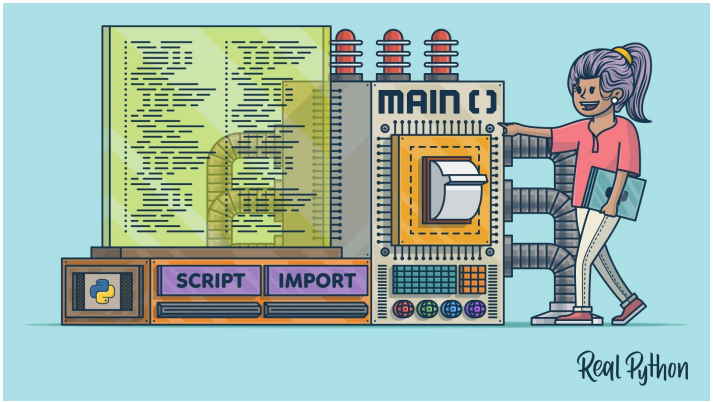
Localização e Adição de Novos

Os módulos/bibliotecas/py são “procurados” da seguinte forma e ordem:

- próprio diretório/pasta
- na variável de ambiente do PYTHONPATH / localização do interpretador PYTHON

Adicionar Módulos Terceirizados

- usar instaladores por terminal (como o pip) que automaticamente instalam o módulo na versão python associada e basta fazer o import com o nome do módulo
- adicionar ao diretório/pasta do ficheiro que vai importá-lo



Módulos

Mais populares

os e os.path Manipulação de arquivos e diretórios

sys Acesso a variáveis e funções usadas interpretador Python no ambiente atual

math Funções matemáticas avançadas

datetime e time Datas, horas e manipulação de tempo

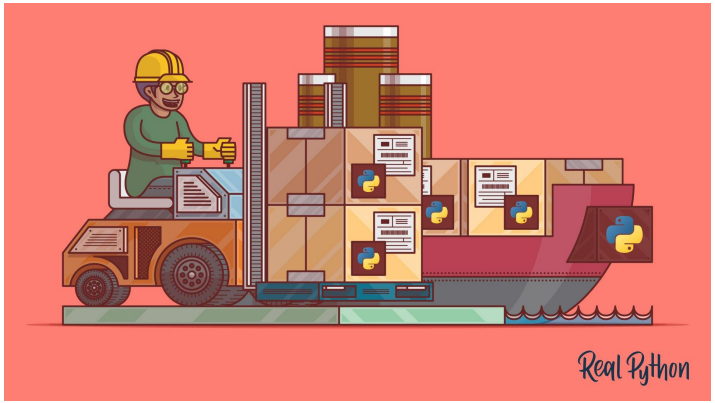
random Gera opções e variáveis aleatórias

json Permite a leitura e escrita de dados no formato JSON

requests Facilita o envio e receção de pedidos HTTP

collections Fornece alternativas avançadas a estruturas de dados

doctest e unittest Bibliotecas para testes unitários automatizados



Dunder Variables/Methods

- Componentes “mágicas” de Python que cumprem determinadas funções necessárias para um ficheiro ser interpretado
- Métodos/variáveis predefinidos de qualquer ficheiro em Python que podem ser personalizáveis
- Identificados por começar e terminar com dois underscores (por exemplo, __init__, __str__, __add__, __doc__, __name__)

Dunder Variable

if __name__ == "__main__":

Contexto

- Forma de garantir que algum código só é executado quando o ficheiro atual é o principal
- No caso de ser importado, assegura que o código só é executado quando algum método ou variável é requisitado
- Sem este método todo o código de um ficheiro iria ser executado ao ser importado para outro ficheiro

```
def add(number_one, number_two):
    return number_one + number_two

if __name__ == "__main__":
    add(3, 5)
```

A Calculadora

que importa

Objetivo

Fazer import do módulo Math nos 3 ficheiros partilhados, de forma a conseguir correr os mesmos ficheiros

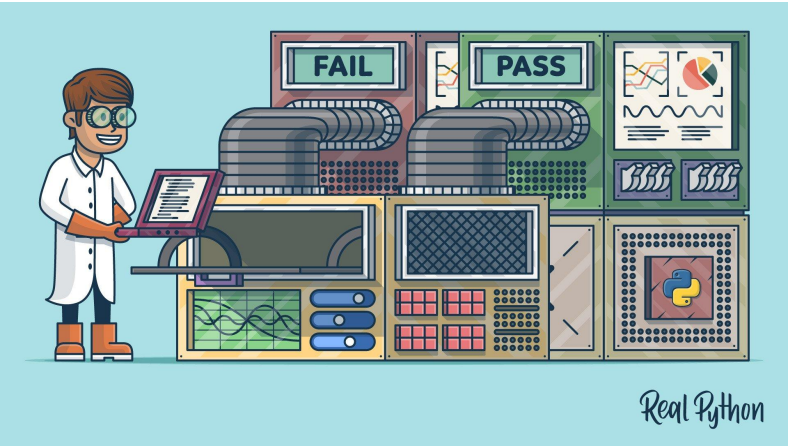
02 unittest e doctest

Testes Unitários

Contextualização

Para garantir a integridade do código através de testes unitários, existem duas frameworks de teste que são bastante populares em Python e que utilizam diferentes abordagens:

- unittest - framework tradicional para testes unitários
- doctest - utiliza a documentação disponível como testes



unittest

Sintaxe

- Utiliza Herança (OOP) para criar novos casos de teste
- Através de chamadas dos métodos sob validação com dados controlados faz-se asserções

Execução

- Pode ser executado pelo terminal a partir do comando python -m unittest nome_do_arquivo_de_teste.py

```
import unittest

def add(a, b):
    return a + b

class TestCalc(unittest.TestCase):
    def test_positive_add(self):
        result = add(3, 5)
        self.assertEqual(result, 8)

if __name__ == '__main__':
    unittest.main()
```

doctest

Sintaxe

- Caso de teste escrito na documentação (docstring) do código.
- Caso de teste identificado pela >>> e resultado esperado colocado na linha seguinte

Execução

- Pode ser executado pelo terminal a partir do comando

  python -m doctest arquivo.py

```
def add(a, b):
    """
    Function to add 2 numbers

    >>> add(2, 3)
    5
    >>> add(0, 0)
    0
    """
    return a + b
```

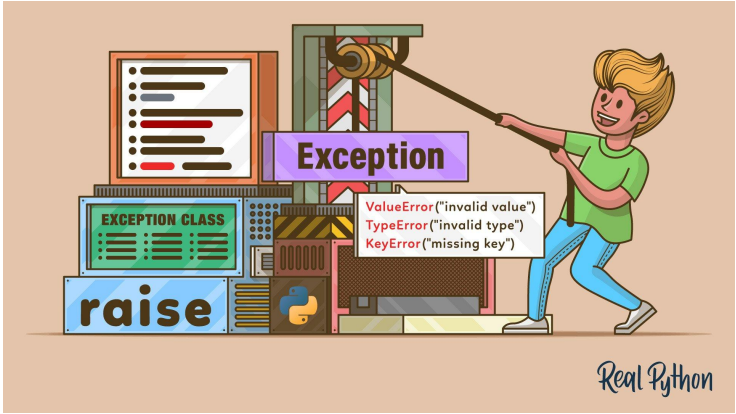
03 Exceções

Exceções

Contextualização

Geral

- São eventos decorrentes de erros e situações inesperadas que ocorrem durante a execução de um programa alterando o correto funcionamento
- Causados por erros de digitação, tentativas de divisão por zero, tentativas de acesso a elementos que não existem em uma lista,...
- Existem estruturas que permitem identificar e tratar esses problemas de maneira controlada, evitando que o programa crashe e proporcionando uma resposta adequada ao evento



Exceções

Sintaxe fundamental

- try Inicia o bloco de código que se pretende executar
- except Captura a exceção e prossegue o tratamento definido

```
try:
    number = int(input("Type a number: "))
except ValueError:
    print("Invalid. Type a number.")
```

Exceções

Sintaxe adicional

- else Ocorre após o except mas apenas se decorreu sem exceções
- finally Ocorre no final do except, e é sempre executado
- raise Permite o envio de exceção personalizada
- except {error} as e Permite utilizar o próprio erro na execução do código

```
def divide(a, b):
    try:
        if b == 0:
            raise AiresDivisionError()
        result = a / b
    except AiresDivisionError as e:
        print(e)
    else:
        print("Result ", result)
    finally:
        print("Division finalized")
```

04 Manipulação de Arquivos

Manipulação de Arquivos

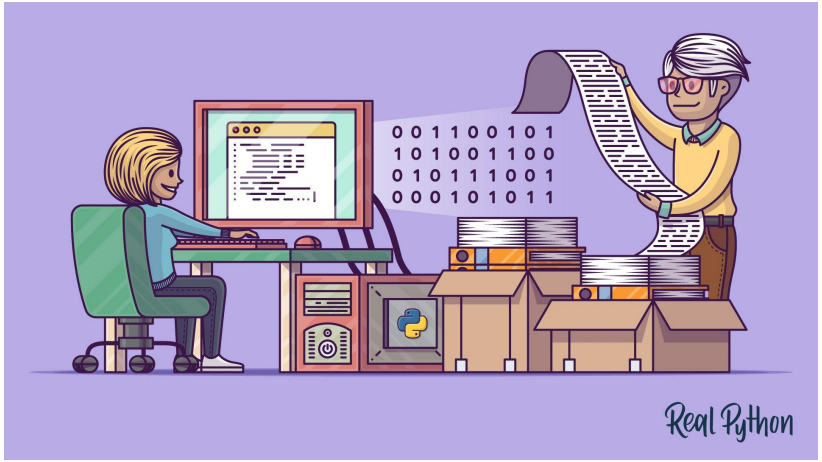
Contextualização

Contexto

- Muitos programas de software precisam de tratar dados locais e/ou salvar dados localmente
- Separação clara entre o código e a base de dados

Python

- with - keyword dedicada a estes casos, seguindo uma dinâmica try/finally



Localização de Arquivo de Dados

Contexto

- Garantir integridade das operações de manipulação de arquivo
- Validar presença de arquivo a ser utilizado

Implementação

- Fazer importação de módulo os
- Usar métodos associados a os.path

```
import os

absolute_path = os.path.abspath("db.txt")
# can use instead relpath() also

if os.path.exists(absolute_path):
    print("Archive available at {absolute_path}")
# will print absolute path if existent
```

Leitura e Escrita de Arquivos

Leitura

Utilizar o bloco with com o método open() com os seguintes argumentos

- path do arquivo
- “r” - identifica que arquivo será para leitura (read)

Por último definir nome da variável associado ao arquivo.

```
with open(“db.txt”, “r”) as db:
    lines = db.readlines()
    for line in lines:
        print(line.strip())
```

Escrita

Utilizar o bloco with com o método open() com os seguintes argumentos

- path do arquivo
- “w” - identifica que arquivo será para leitura (write)

Por último definir nome da variável associado ao arquivo.

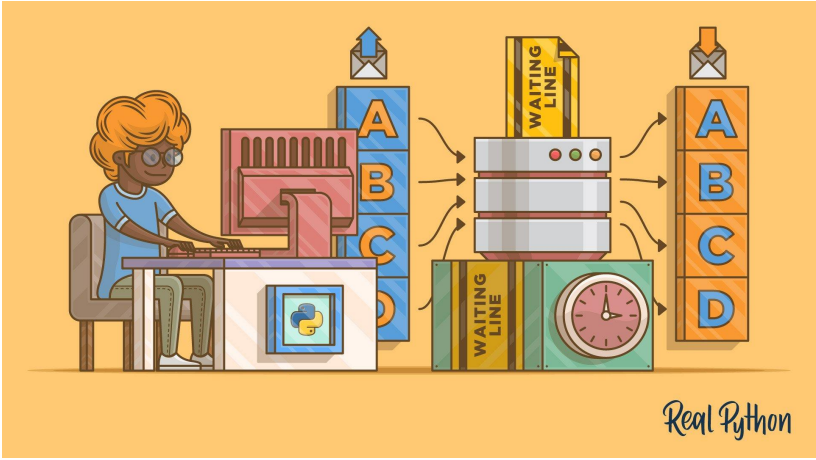
```
with open(“new_db.txt”, “w”) as db:
    db.write("Random Data")
```

E não é muitos, e não são poucos, não é? Bastantes...

Contador de Palavras

Objetivo

Garantir que o ficheiro do tipo txt é lido pelo programa, para proceder à contagem de palavras.



Rumos

Formação e Certificação Técnica que potenciam **Profissionais e Organizações**

Lisboa

Edifício Mirage – Entrecampos
Rua Dr. Eduardo Neves, 3
1050-077 Lisboa

ver google maps ↗

Tel +351 217 824 100
Email info@training.rumos.pt

Porto

Edifício Mirage – Entrecampos
Rua Dr. Eduardo Neves, 3
1050-077 Lisboa

ver google maps ↗

Tel +351 222 006 551
Email info@training.rumos.pt

Siga-nos

Rumos