

Departamento de Engenharia de Sistemas e Computação

Módulos e Exceções

Uso de math e random; tratamento de erros com try/except

João Araujo

Algoritmos Computacionais I

Aula 13

Roteiro

- Prática com os módulos `math` e `random` (Cap. 4.2)
- Simulações com números aleatórios
- Erros em tempo de execução: o que são *exceções*
- `try/except`: tratando erros (4.7)
- Programas robustos: validar entradas do usuário

O módulo `math`

```
import math

print(math.pi, math.e)                  # constantes
print(math.sqrt(2), math.pow(2, 10))    # raiz, potência
print(math.sin(math.radians(30)))       # seno (em radianos!)
print(math.log(100, 10), math.log10(1000), math.exp(1))
print(math.floor(3.7), math.ceil(3.2), math.fabs(-4))
print(math.hypot(3, 4))                 # 5.0
print(math.degrees(math.pi))            # 180.0
```

- Funções trigonométricas usam radianos: converta com `radians()`
- Exercício 4.4: explore outras funções no manual (`factorial`, `gcd`, `isclose`...)

Exercícios com `math`

Exercício 4.2: tabela de raízes quadradas

```
import math
for n in range(1, 11):
    print(n, math.sqrt(n))
```

Exercício 4.3: quadrado e cubo

```
n = int(input("Número: "))
print(n, n**2, n**3)
```

Para pensar

`math.isclose(0.1 + 0.2, 0.3)` devolve `True`. Por que é melhor que `==`? (Veja a Aula 2.)

O módulo `random`

```
import random

print(random.random())            # float em [0, 1)
print(random.randint(1, 6))       # inteiro entre 1 e 6 (inclusive)
print(random.uniform(10, 20))     # float entre 10 e 20
print(random.choice(['a', 'b', 'c']))
random.seed(42)                   # reprodutibilidade
```

- Números *pseudoaleatórios*: gerados por uma fórmula; com a mesma seed a sequência se repete
- Úteis em simulações, jogos e testes de programas
- Exercício 4.13: simular o lançamento de um dado de 6 lados

Simulando 1000 lançamentos de um dado

```
import random

contagem = 0
total = 1000
for i in range(total):
    dado = random.randint(1, 6)
    if dado == 6:
        contagem += 1
print(f"Saiu 6 em {contagem} de {total} lançamentos")
print(f"Frequência: {contagem/total:.3f} (esperado: {1/6:.3f})")
```

- Lei dos grandes números: quanto mais lançamentos, mais perto de $1/6$
- O mesmo princípio estima π no programa Monte Carlo da Aula 11

Fonte: Exercício 4.13; code/Cap4/prog.4.23_calcula_pi_main.py

Quando o programa quebra

```
x = int(input("Numerador: "))
y = int(input("Denominador: "))
print(x / y)
```

```
Numerador: abc
ValueError: invalid literal for int() with base 10: 'abc'

Numerador: 5
Denominador: 0
ZeroDivisionError: division by zero
```

- Erros **de sintaxe** são detectados antes de executar; erros **em execução** geram uma exceção
- Sem tratamento, o programa *termina* com mensagem técnica; o usuário não deveria ver isso

try/except

```
def divide(x, y):
    """Divide x por y."""
    try:
        z = x / y
        print(z)
    except ZeroDivisionError:
        print('Tentativa de divisão por zero')

divide(1, 2)       # 0.5
divide(1, 0)       # Tentativa de divisão por zero
```

- O bloco `try` é executado; se ocorrer a exceção citada, o fluxo *salta* para o `except`
- Depois do tratamento o programa continua normalmente

Fonte: `code/Cap4/prog.4.25_divide_zero.py`

Várias exceções

```
def divide():
    """Divide x por y."""
    try:
        x = int(input('Digite um numerador:'))
        y = int(input('Digite um denominador:'))
        print('A divisão de', x, 'por', y, 'é igual a', x/y)
    except ZeroDivisionError:
        print('Tentativa de divisão por zero.')
    except ValueError:
        print('Número inválido.')

divide()
```

Cada except trata um tipo diferente de erro; capture só o que sabe tratar (evite except: sem tipo).

Fonte: code/Cap4/prog.4.26_divide_exceto.py (Exercícios 4.16 e 4.17)

Laço de validação com exceção

```
def main():
    while True:
        try:
            num = int(input('Quantos pontos devem ser sorteados? '))
            break
        except ValueError:
            print('Erro: Somente números são aceitos.')
    pi, erro = calcula_pi(num)
    print('Com', num, 'pontos, o valor de pi é', pi,
          'com erro', erro)
```

- Padrão clássico: `while True` + `try` + `break` ao dar certo
- O usuário só avança quando a entrada é válida

Fonte: `code/Cap4/prog.4.24_calcula_pi_try.py`

else e finally

```
try:
    valor = float(input("Valor: "))
except ValueError:
    print("Não é um número.")
else:
    print("Você digitou", valor)       # só se NÃO houve exceção
finally:
    print("Fim da leitura.")           # sempre executa
```

- `else`: código que depende do sucesso do `try`
- `finally`: limpeza obrigatória (por exemplo, fechar um arquivo: Aula 23)
- Exceções comuns: `ValueError`, `ZeroDivisionError`, `IndexError`, `KeyError`, `FileNotFoundError`, `TypeError`

Atividades

1. **Exercício 4.16:** crie uma lista com 3 elementos e peça ao usuário um índice; mostre o elemento. Capture `IndexError` (índice fora da lista).
2. **Exercício 4.17:** acrescente a captura de `ValueError` (digitou texto).
3. Escreva `le_inteiro(mensagem)`: pede até o usuário digitar um inteiro válido e o devolve. Use-a em seus programas de agora em diante!
4. Simule 10 000 lançamentos de dois dados e mostre a frequência de cada soma (2 a 12). Qual é a mais frequente?

Para praticar: exercícios do livro

Site do livro – Capítulo 4

- Módulos: **4.1 a 4.4**, **4.13** (dado de 6 lados)
- Exceções: **4.16, 4.17**
- Veja também **5.3** (Fahrenheit com `try/except`) no Capítulo 5

Enunciados e soluções comentadas em
`https://www.araujo.eng.uerj.br/livro/exercicios/cap4/`

Exercícios adicionais (sem solução): `https://www.araujo.eng.uerj.br/livro/extras/`

Resumo da aula

- `math`: constantes e funções matemáticas (ângulos em radianos)
- `random`: números pseudoaleatórios para simulações
- Exceção = erro em tempo de execução; `try`/`except` impede o programa de "quebrar"
- Capture o tipo de erro que você sabe tratar
- Próxima aula: revisão geral dos capítulos 1 a 4