Programação II
+
Estruturas de Dados para Bioinformática

Ambiente de desenvolvimento e conceitos base

Hugo Pacheco

DCC/FCUP
22/23

Conceitos base

- As primeiras aulas teóricas vão ser de revisão dos conceitos base de programação em Python

- Já devem ser bastante familiares…

- Vamos seguir em grande parte a apresentação da bibliografia principal:

  - Allen Downey; How to think like a computer scientist. ISBN: 0-9716775-0-6

Programação

- Programação ⊃
  - **Matemática**: linguagens formais para expressar programas.
  - **Engenharia**: desenhar e construir sistemas compondo componentes, avaliar alternativas.
  - **Ciências**: observar comportamento de sistemas complexos, formular hipóteses, testar previsões
- Competências ⊃
  - **Resolução de problemas**: formular problemas, encontrar e exprimir soluções
  - Abstração: generalizar, identificar o essencial do problema
  - Decomposição: separar problemas complexos em sub-problemas mais simples
  - Reconhecimento de padrões: identificar similaridades e reutilizar soluções

A linguagem Python

- linguagem de alto nível
    - mais próxima do raciocínio abstrato, matemático e longe de como é executada por um computador
        - programas mais curtos, sintaxe mais legível, menos erros, maior portabilidade
- extremamente popular, especialmente entre não-programadores; forte comunidade e suporte
- muitas bibliotecas para ciência de dados (NumPy, SciPy, Pandas,…) e visualização (Matplotlib, Seaborn, Plotly,…)

A linguagem Python

- Interpretador: observar o resultado imediato de computações
    - vários interpretadores no browser, e.g., Python.org
- Compilador: converter todo um programa numa aplicação executável
- Vários compiladores no browser, e.g., W3Schools
- Tipos dinâmicos, gestão de memória automática
- Múltiplos paradigmas: imperativo, funcional, orientado a objetos
- Extensas bibliotecas e documentação
- Vários Integrated Development Environments (IDEs), e.g., PyCharm
    - Vamos (poder) utilizar o IDE online replit, e.g., Projeto 0.

Um programa Python

```
print("Hello, World!")   # Disse Olá Mundo!
print(type("Hello, World!"))

print(42000)
print(type(5))
print(type(3.2))
print(type("3.2"))

import datetime
print(datetime.datetime.now())
```

Corra este programa num interpretador e num compilador

Erros

- Programação é um processo complexo, suscetível a erros humanos
- Tipos de erros:
  - **Sintaxe**: linguagem não suportada

```
2 + / 4
```

  - **Execução**: comportamento não suportado

```
'as' + 3
```

```
2 / 0
```

  - **Semântico**: comportamento indesejado

```
2 + 4 * 3
```

vs

```
(2 + 4) * 3
```

Explore as mensagens de erro exatas

Erros de tipos? 

- Podemos anotar termos variáveis/funções com tipos
- Relembrar que Python é uma linguagem com tipagem dinâmica

```
def add(x:int,y:str) -> list[int]:
    return x + y

add(1,2)
```

- Algum erro? 

Debugging

- Depuração de erros é uma das principais competências de um programador

- Ao lidar com bibliotecas avançadas, vamos ter que aprender a navegar por vários tipos de erros

- Uma ciência experimental

    - lançar hipótese sobre o que está errado, alterar o programa de acordo com a hipótese e tentar de novo

    - hipótese correta: um passo em frente para a versão final

    - hipótese errada: encontrar nova hipótese

Debugger

- Auxilia a depuração de programas, permitindo:
  - executar instrução-a-instrução
  - definir breakpoints
  - visualizar estado interno
  - executar expressões sobre estado interno
- Poor man’s debugger: inserir prints cuidadosamente no código
- Debugger visual no browser, e.g., PythonTutor
- Debugger no IDE, e.g., PyCharm

Um programa com um erro

```
message = "Ola"
message = “Mundo"
print(message)
print(type(message))

a = 0
b = 0          # a e b são agora iguais
print(a, b)
a = 3          # a e b não são mais iguais
print(a, b)
a = a / b      # a toma um valor inválido
print(a)
```

Depure este programa no PythonTutor e no PyCharm

Documentação

- Documentação da standard library disponível em https://docs.python.org/ (listagem completa)

- Documentação de biblioteca disponível na página da mesma, e.g., NumPy (https://numpy.org/doc/)

- Excelente para descobrir em detalhe que funcionalidade oferece cada módulo, e como utilizar cada função ou método

- Não é ideal para exemplos ou começar a aprender

Documentação

- Meta-notação que descreve a sintaxe do Python, sem fazer parte dela
  - elementos facultativos entre [parêntesis retos]
  - palavras reservadas em **bold**
  - variáveis em *itálico*
- *E.g., documentação de* strings

Explore a documentação através do PyCharm

Documentação

- É boa prática definir documentação de funções

```
def sum_subtract(a, b, operation="sum"):
    """
        Sums os subtracts two numbers
        :param a -- the first number
        :param b -- the second number
        :returns a + b or a - b
        """
    if operation == "sum":
        return a + b
    elif operation == "subtract":
        return a - b
    else:
        print("Incorrect operation.")

print(sum_subtract(1, 2))
```

- Não suportado para variáveis, utilizar comentários

```
# constante com o valor de pi
pi : float = 3.14
```

Outros recursos

- Tutoriais:
  - https://realpython.com/
  - https://www.learnpython.org/
  - https://www.w3schools.com/python/
  - https://wiki.python.org/moin/BeginnersGuide/Programmers
- Questões gerais:
  - Teams da cadeira (recomendado)
  - stackoverflow, reddit (cautela)

Conversão de tipos

```
int(3.14)
int(3.9999) # doesn't round to the closest int!
round(3.9999)
int(-3.999) # result is closer to zero
minutes = 120
int(minutes / 60)
int("2345") # parse a string to produce an int
int(17) # even works if already an int
int("23 bottles")
float(17)
float("123.45")
str(17)
str(123.45)
```

Inteiros

```
print(7 / 4)        # division
print(7 // 4)       # integer division

total_secs = int(input("How many seconds? "))
hours = total_secs // 3600
rem_secs = total_secs % 3600
minutes = rem_secs // 60
fin_secs = rem_secs % 60
print("Hrs=", hours, "mins=", minutes,
"secs=", fin_secs)

print(math.log(3**2))
```

Strings

```
message = "Hi There"
print(message - 1)

print("Hello" / 123)
print(message * "Hello")
print("15" + 2)

print(message + " " + "John Snow")
print(message * 3)

message = message.casefold()
message = message.capitalize()
print(len(message))
```

Composição

```
# first without composition
response = input("What is your radius? ")
r = float(response)
area = 3.14159 * r**2
print("The area is", area)
```

```
# now let's use composition
r = float( input("What is your radius? ") )
print("The area is", 3.14159 * r**2)

# All in one statement
print("The area is", 3.14159*float(input("What is
your radius? " ))**2)
```

Lambda

- Função anónima declarada no momento em que é usada
- As seguintes definições são todas equivalentes

```
print(x+10)

print((lambda a : a + 10)(5))

plus10 = lambda a : a + 10
print(plus10(5))

def plus10(x): return x + 10
print(plus10(x))
```