

# **Estruturas de Dados para Bioinformática**

Hugo Pacheco

DCC/FCUP  
24/25

# Apresentação da cadeira

# Programação

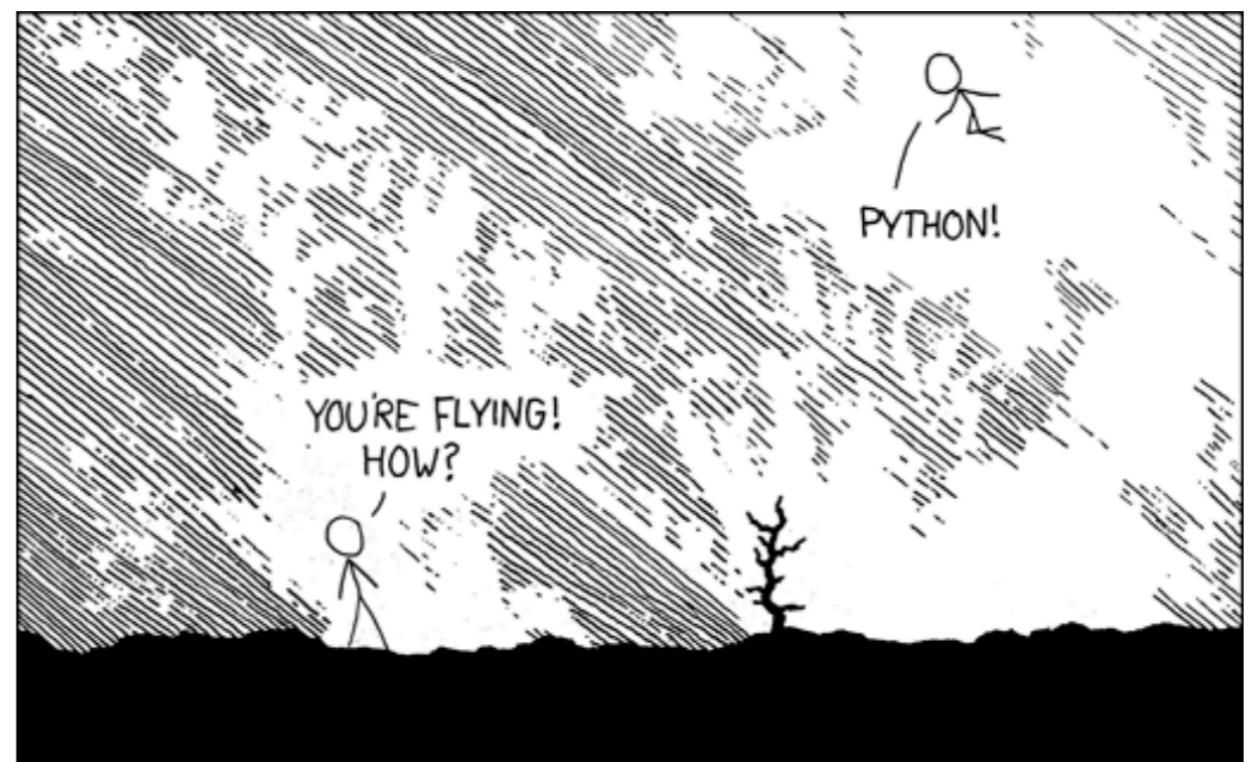
- Programação =
  - Automatização de processos
  - Resolução de problemas
  - Extração de informação
  - Visualização de informação

# Programação 101

- Introdução à programação e aos computadores
- Conceitos base de programação em Python
- Apresentação mais **bem-fundada**:
  - Partir dos fundamentos de programação: funções, listas, dicionários, classes, exceções, etc
  - Escrever programas simples que ilustram os conceitos
  - Aprender programação como resolução de problemas

# Python

- Uma linguagem de alto nível e de alta produtividade
- “batteries included”: milhares de bibliotecas prontamente disponíveis
  - gráficos
  - processamento de dados
  - web
  - domínio específico (estatística, geografia, biologia, etc)
  - ...

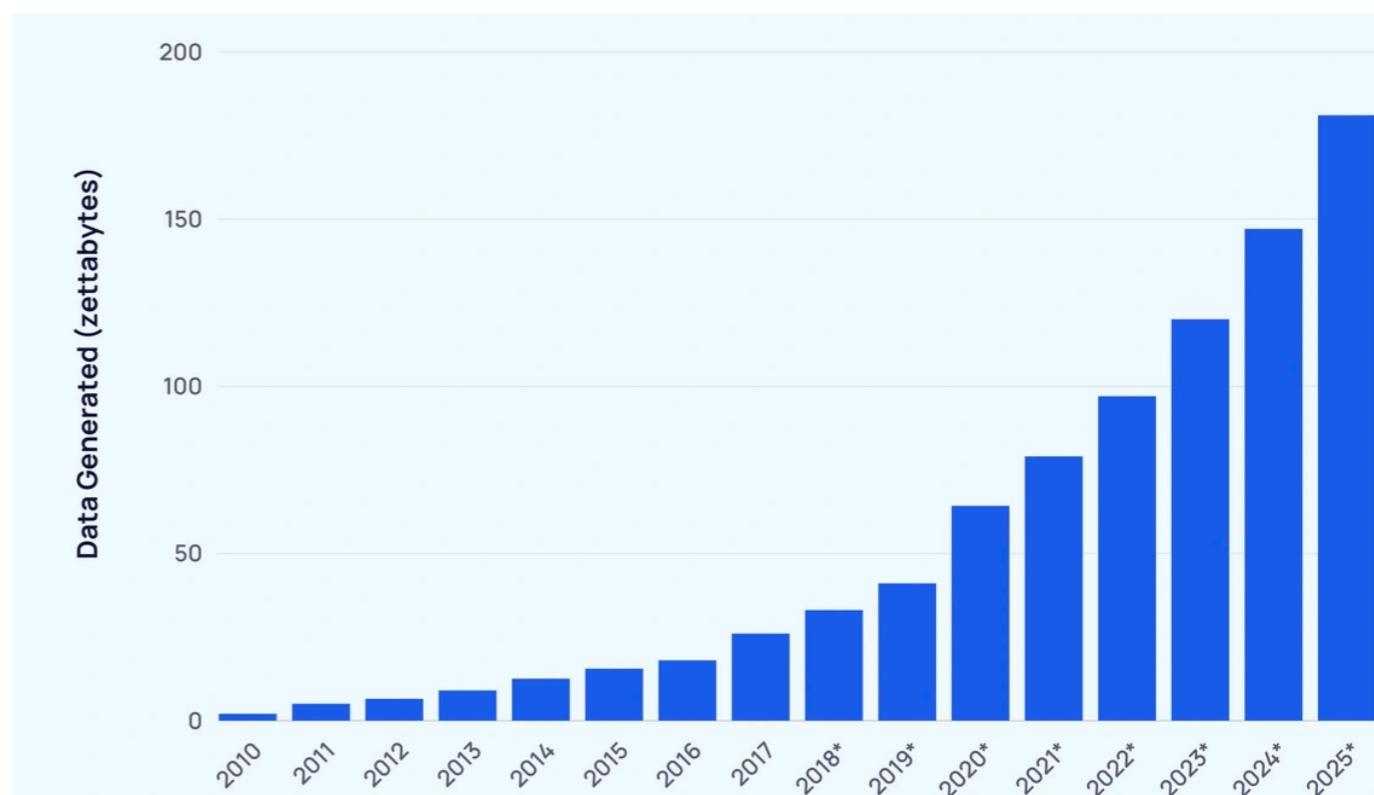


# Esta cadeira

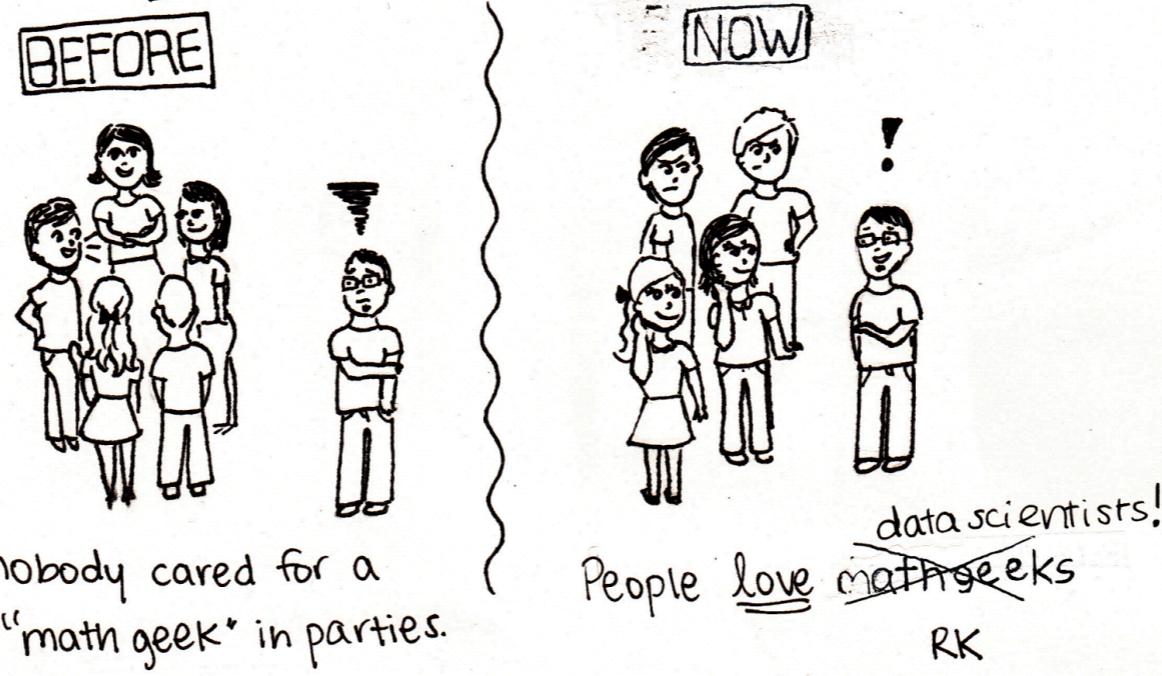
- Cadeira profundamente prática de programação em Python
- Apresentação mais **pragmática**:
  - Aprender a utilizar alguns dos melhores recursos de programação para manipulação automática de informação
  - Desenvolver familiaridade com as bibliotecas e ambiente de desenvolvimento existentes e melhorar experiência de programação
  - O objetivo não é formar especialistas em Ciências da Computação. Tanto quanto possível, não vamos nem queremos olhar para dentro do “capô”

# Análise de dados

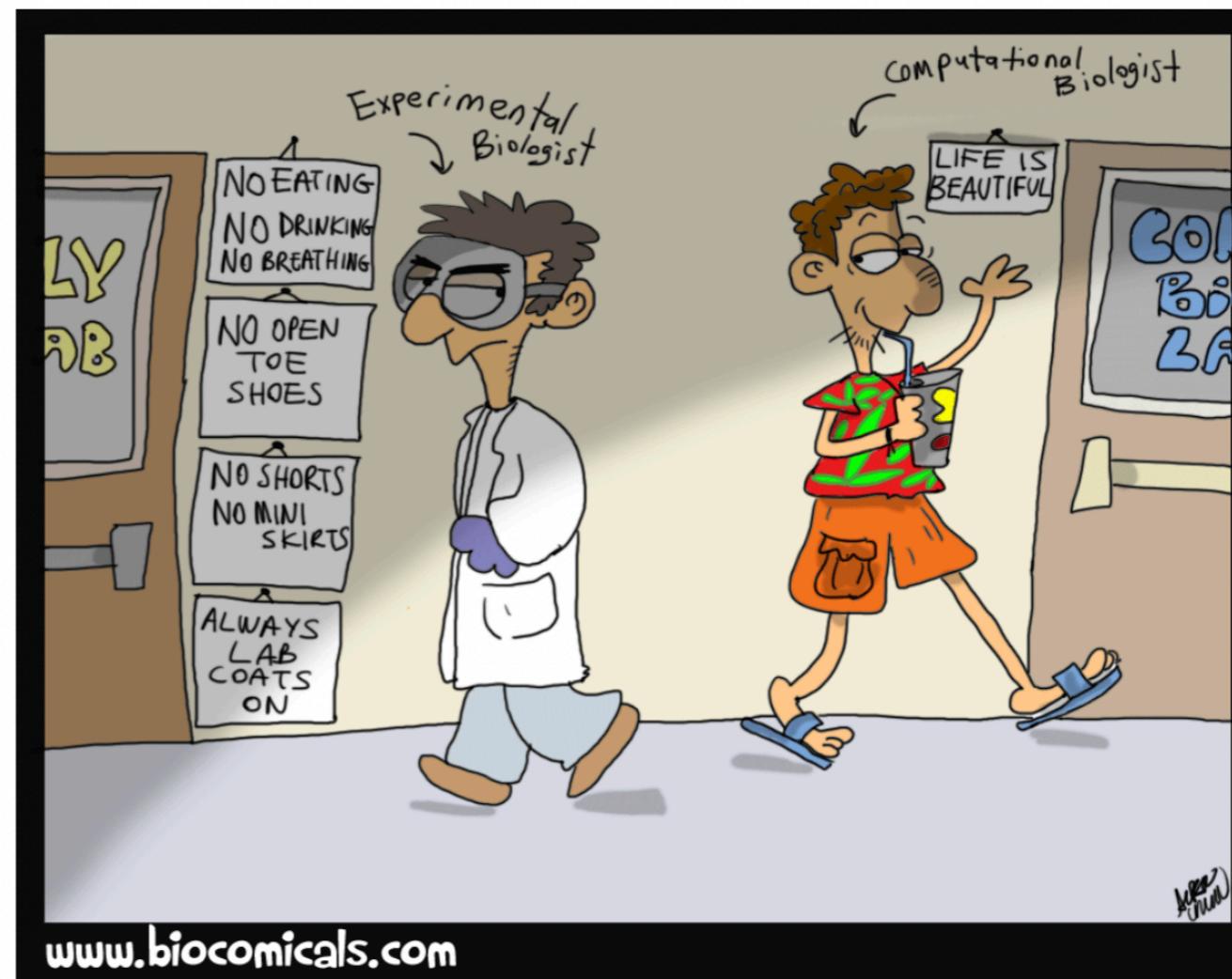
- É uma fração significativa de toda a programação que é feita hoje em dia
- É a principal forma de programação feita por programadores não profissionais
- Python é uma linguagem bastante forte neste domínio
- Pode ser aplicada em domínios variados, não necessariamente por especialistas de informática: biologia, matemática, física, ciências sociais, medicina, geografia, economia, etc
- “Data-driven world”: grandes quantidades de dados, necessidade de automatização e extração de conhecimento



# The Rise of Data Scientists



IN CS, IT CAN BE HARD TO EXPLAIN  
THE DIFFERENCE BETWEEN THE EASY  
AND THE VIRTUALLY IMPOSSIBLE.



# Programa

## 1. Revisão dos conceitos base

- números, strings, listas, dicionários, classes, etc

## 2. Análise de dados

- formatos comuns de partilha de informação
- modelos comuns de representação e manipulação de dados

## 3. Visualização de dados

- diferentes formas de visualização de dados (gráficos, mapas)
- Construção de animações para visualização de dados

# Avaliação

- Não serão registadas presenças (teóricas e práticas)
- Avaliação completamente prática
- **70% nota prática:**
  - 3 trabalhos práticos **em grupos de 2 elementos** para serem desenvolvidos nas aulas práticas e em casa
- **30% nota oral:** defesa do trabalho no fim do semestre
  - elegância, organização e documentação do código
  - capacidade de resposta a perguntas sobre o trabalho
  - componente contínua (o docente acompanhou o vosso trabalho nas aulas práticas)
- É possível melhorar e entregar os trabalhos em época de recurso, mas será exigida nova funcionalidade; têm que ter frequência (ter entregue os trabalhos) em época normal

# Trabalhos práticos

- O enunciado de cada trabalho estará dividido em Tarefas.
- Podem utilizar os enunciados de anos anteriores como exercícios adicionais para treinar
- Cotações de cada Tarefa serão decididas mais à frente consoante o progresso das aulas

# Trabalhos práticos

## 1. Análise de texto (**ano anterior, TBD**)

- processamento do texto de uma obra conhecida: contar palavras, etc
- análise simples de sequências de DNA: encontrar e separar proteínas

## 2. Análise de dados (**ano anterior, TBD**)

- análise de dados fornecidos pela Nobel Foundation
- processamento de imagens de organismos retirados da Cell Image Library

## 3. Visualização e animação de dados (**ano anterior, TBD**)

- Construção e visualização de fractais
- Análise e visualização de datasets vários
- Visualização de trophic networks

# Entregas

- Entrega por email para o docente ([hpacheco@fc.up.pt](mailto:hpacheco@fc.up.pt))
  - Enviar apenas um ficheiro de código com o nome projeto?.py
  - Não enviar ficheiros de suporte (texto, json, imagens, gráficos, ...)
  - Não alterar o template (assinaturas de funções e código pré-definido)
- Datas **provisórias**, podem vir a ser alteradas com o calendário
  - Projeto 1: entrega até 23 Março
  - Projeto 2: entrega até 13 Abril
  - Projeto 3: entrega até 01 Junho

# Informações Gerais

- Sigarra: site oficial da cadeira
- Github: repositório com todo o material de suporte à cadeira (slides, projetos práticos, etc)
- MS Teams: plataforma para comunicação fora das as aulas; instant messaging
- Docente: Hugo Pacheco ([hpacheco@fc.up.pt](mailto:hpacheco@fc.up.pt))
- Não existe horário de atendimento pelo menos por enquanto. Apela-se à comunicação pelo MS Teams ou por email institucional

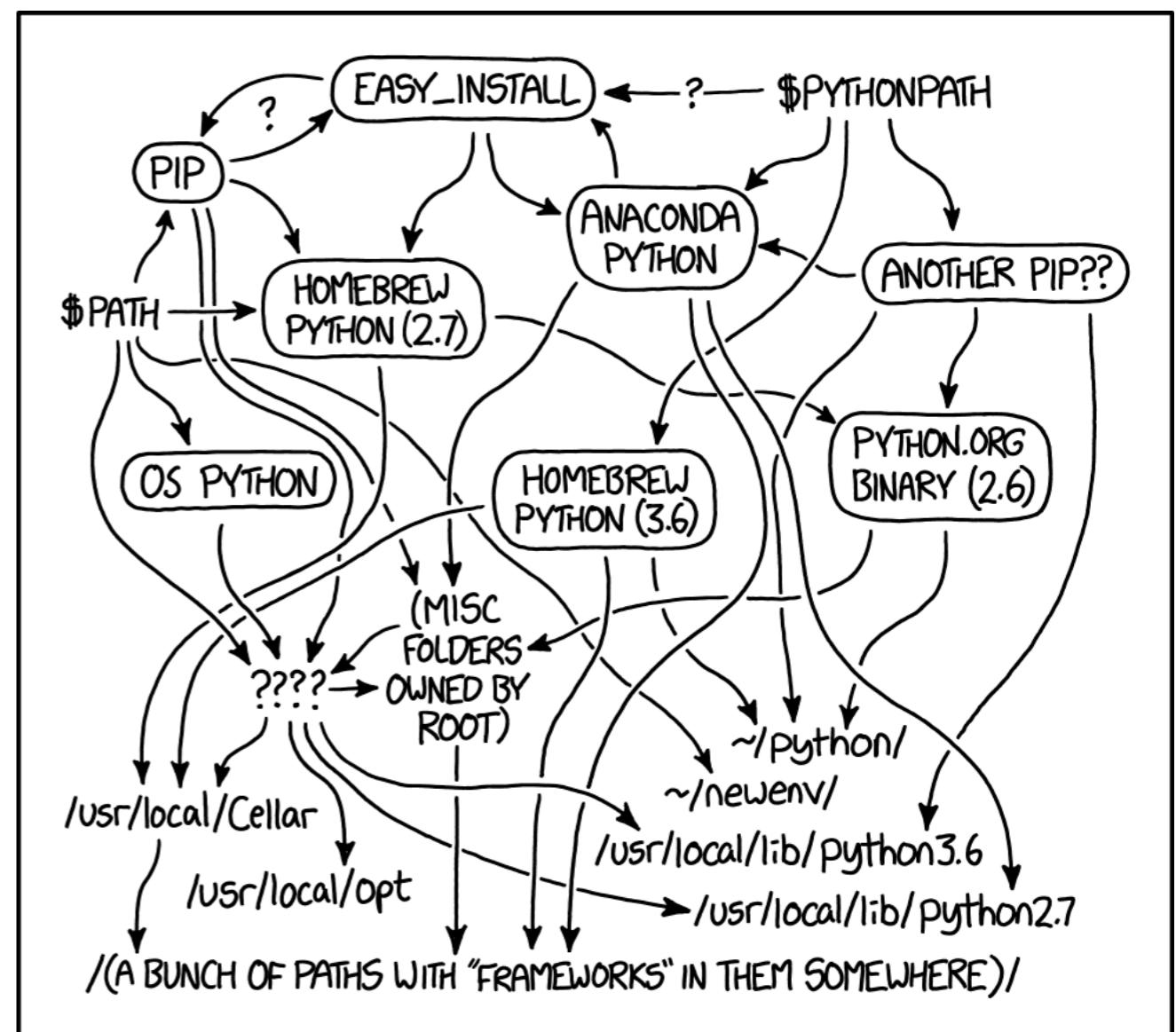
# Aulas

- Em regime presencial; não serão contabilizadas faltas
- Uma só aula teórico-prática
  - Teórica: apresentação da matéria e exemplos práticos de utilização
  - Prática: resolução dos projetos práticos



# TPC: Instalar o Python

- Existem muitas formas de instalar o Python e bibliotecas associadas: *binário, anaconda, pip, brew, apt-get, etc*
- Se já estão familiarizados com um ambiente, usem esse; evitem ter múltiplas instalações do Python



# TPC: Instalar o Python

- Como sugestão...
- Instalar Python 3.x “nativo”
  - Windows: fazer download do instalador oficial em <https://www.python.org/downloads/>
  - Linux: *sudo apt-get install python3*
  - Mac OS: *brew install python3*
- Utilizar um IDE como o PyCharm (<https://www.jetbrains.com/pycharm/download/>) ou o VSCode (<https://code.visualstudio.com/Download>)
  - code highlighting, debugging, documentação, etc
  - gestão e instalação automática de pacotes, não interfere com outras possíveis instalações

# Projeto 0 - Desenferrujar

- Primeiras duas semanas de aulas práticas para rever matéria e aferir conhecimentos
- Enunciado do Projeto 0
- Para as aulas práticas:
  - Instalem o Python + IDE
  - Há algumas alternativas web, e.g.,
    - Formem grupos (notificar docente)



# Ambiente de desenvolvimento e conceitos base

# Conceitos base

- As primeiras aulas teóricas vão ser de revisão dos conceitos base de programação em Python
- Já devem ser bastante familiares...
- Na revisão, vamos seguir em grande parte a apresentação da bibliografia:
  - 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](#) ou [VSCode](#)

# 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

# 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)
```

# Funções

- Um dos elementos principais em programação é decomposição de problemas, e.g. decomposição em funções
- Analogia com funções matemáticas, recebem inputs e retornam outputs
- E.g., segunda lei de Newton para queda de corpos em função da velocidade inicial  $v_0$  e tempo  $t$ , com constante gravítica  $g$

```
def y(v0, t):  
    g = 9.8  
    return v0 * t - 1 / 2 * g * (t**2)  
help(y)  
y(1, 2)
```

# Funções (composição)

- Funções podem ser compostas, i.e., chamar outras funções
- Funções podem definir sub-funções locais (não recomendado)
- E.g., calcular a área de um círculo com raio igual à distância entre dois pontos

```
import math
def distance(x1, y1, x2, y2):
    return math.sqrt( (x2-x1)**2 + (y2-y1)**2 )
def area(radius):
    return math.pi * radius**2
def area_of_circle(xc, yc, xp, yp):
    return area(distance(xc, yc, xp, yp))
print(area_of_circle(0, 0, 0, 1))
```

# 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))
```

# 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 num IDE

# Erros de tipos?



- Podemos anotar 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 = "Olá"  
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 IDE

# Documentação

- É boa prática definir documentação de funções

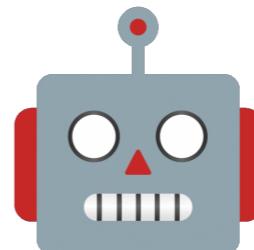
```
def sum_subtract(a, b, operation="sum"):  
    """  
        Sums or subtracts two numbers  
        :param a -- the first number  
        :param b -- the second number  
        :returns a + b or a - b  
    """  
  
    if operation == "sum":  
        return a + b  
    elif operation == "subtract":  
        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:
  - recomendado: Teams da cadeira
  - cautela: [stackoverflow](#), [reddit](#), [ChatGPT!?](#)



# Inteligência Artificial!?



- ChatGPT = chat livre, linguagem natural
  - Interpreta enunciados e produz código; eficaz apenas em questões simples ou bastante comuns
- GitHub Copilot (grátis c/  
Github Education), Google Gemini ou Replit AI = assistentes de programação
  - Preenche código, melhor noção de contexto
  - Gera explicações/ comentários de código
- O que sobra para nós?

