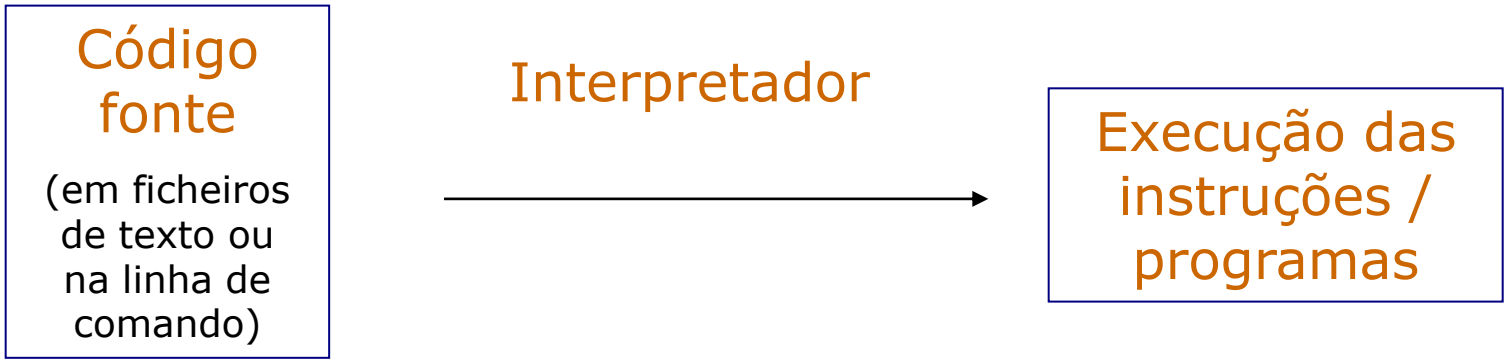
NOÇÕES BÁSICAS DE PROGRAMAÇÃO EM PYTHON

Edição e execução dos programas

- **Editor –** permite escrever o código fonte dos programas, podendo ter ferramentas que auxiliem na tarefa (e.g. highlight de sintaxe)
- **Linha de comandos (shell)** – permite executar instruções individuais e ver os seus resultados interactivamente
- **Interpretador** – aplicação que interpreta o código fonte escrito na shell ou em programas, executando-o
- **IDE** – ambiente integrado de desenvolvimento: inclui os anteriores e outras ferramentas de suporte à programação



As “nossas” linguagens de trabalho

Vamos utilizar, como **Linguagem de Programação**, a linguagem **Python** – versão 3.5

Vamos escrever os programas num editor de texto ou ambiente integrado desenvolvimento (IDE)

Teremos disponível (no IDE ou na linha de comando) um interpretador para poder executar instruções linha a linha (na linha de comandos) ou programas completos

Instalação do ambiente de trabalho

- Requisitos obrigatórios:
  - Python: www.python.org - *versão 3.5.x*

    (documentação: docs.python.org/3)
  - *BioPython*
- Opcional, mas muito útil, uso de um IDE
  - Recomendado: ***Anaconda*** (inclui editor *Spyder*, linha de comando otimizada, *Jupyter notebooks*, gestor de *packages*, …); fácil de instalar; livre; todos os sistemas
    - *Download, instalação, documentação: http://www.continuum.io/anaconda*

Conceitos importantes

- A linguagem Python baseia-se em vários tipos de entidades:
  - **VARIÁVEIS ou OBJETOS** – guardam dados de diversos tipos (primitivos do Python ou definidos pelo programador) dando-lhes um **nome;**
  - **FUNÇÕES** – usadas para realizar o processamento de dados, bem como as operações de input/ output. Em python podem ser ou não aplicadas sobre objetos
  - **PROGRAMAS (scripts)** – tipicamente um conjunto de instruções incluindo chamadas de funções com um fluxo (ordem) definido para a resolução de uma (ou várias) tarefas

Método de trabalho

- Fase 1: vamos explorar a linha de comandos (**shell**) interativa do python para executar instruções simples e explorar funções pré-definidas
- Fase 2: vamos elaborar **programas**/ ***scripts*** simples com uma sequência de instruções bem definida incluindo entrada e saída de dados; usaremos tipos primitivos e funções pré-definidas do python
- Fase 3: exploraremos as potencialidades do Python na definição de novas **funções** (permite reutilização de código; programas mais complexos)
- Fase 4: usaremos o python como uma linguagem de **programação por objetos** na definição de novas classes (melhor organização dos programas e reutilização mais avançada de código)

Variáveis e objetos

- **Variáveis** são entidades (nomes) que podem referenciar valores distintos durante a execução de um programa
- Variáveis podem guardar valores "simples" de tipos pré-definidos: valores **numéricos** (inteiros ou decimais), valores **lógicos** (verdadeiro ou falso), **strings**
- O python não define os tipos das variáveis explicitamente, sendo "assumidos" pelo contexto da computação. Isto facilita a sintaxe mas pode causar erros de execução com mais facilidade (e.g. variáveis com o mesmo nome; erros na escrita do nome da variável)
- Variáveis podem também guardar dados mais complexos. Em Python, estes podem ser implementados por **objetos** que constituem instâncias de diversas **classes**, pré-definidas ou programadas pelo utilizador

Funções

- Permitem realizar operações de processamento de dados

Argumentos da função: entradas

Resultado: saída

Embora sendo mais flexíveis, são semelhantes a funções matemáticas, recebem valores (variáveis) de entrada e retornam (opcionalmente) um dado resultado

Funções

- Existem funções pré-definidas no python, podendo também definir-se novas funções (como iremos ver mais tarde ...)

- **Invocação**: permite executar funções
  - invoca-se a função passando os argumentos de entrada na forma ***função(argumentos)*** e obtendo-se o resultado.
  - algumas funções podem ser aplicadas sobre um objeto, usando-se a notação ***objeto.função(argumentos)***
  - neste último caso, o conteúdo do objeto sobre o qual se aplica a função pode ser alterado

Variáveis: operação de **atribuição**

- Através da atribuição associa-se um nome de uma variável a um valor
- Sintaxe do python: **nome_variavel = valor** (note-se que para testar se uma variável é igual a um valor se usa o operador ==).
- Exemplo: **a = 5** significa que o valor da variável de nome *a* passa a ser de *5*

Variáveis numéricas

- Guardam valores **inteiros** ou **reais** dentro de uma determinada gama
- Aos seus valores podem aplicar-se operadores aritméticos: +, -, *, /, exponenciação (**), divisão inteira(//), resto da divisão inteira (%)
- Tipicamente, existem também funções pré-definidas:
  - Módulo (abs)
  - Arredondar (round)
- Package math inclui um numeroso lote de funções matemáticas e científicas úteis
  - Exemplos: sin, tan, cos, sqrt, exp, factorial, log, tanh, ...

Variáveis numéricas: exemplos

```
>>> x = 5
>>> y = 4
>>> x + y
9
>>> x * y
20
>>> x / y
1.25
>>> x // y
1
>>> z = 25
>>> z % x
0
>>> z % y
1
>>> x ** y
625
```

```
>>> abs(-3)
3
>>> round(3.4)
3.0
>>> float(2)
2.0
>>> int(3.4)
3
>>> int(4.6)
4
>>> int (-2.3)
-2
>>> 0.00000000000001
1e-14
>>> 2.3e-3
0.0023
```

```
>>> import math
>>> math.sqrt(4)
2.0
>>> math.sin(0.5)
0.479425538604203
>>> math.log(x)
1.6094379124341003
>>> math.pi
3.141592653589793
>>> math.tan(math.pi)
-1.224646799147353e-16
>>> math.e
2.718281828459045
>>> math.exp(1)
2.718281828459045
>>> math.log(math.e)
1.0
```

Variáveis do tipo **list**

- Permitem armazenar e processar sequências (ordenadas) de valores
- Definir uma variável x do tipo list: **x = [1,2,3,4,5]**
- Aceder a uma posição/ modificar o seu valor:

  1ª posição: **x[0] = 10** (primeira posição tem índice 0)

  elem = x[3] atribuir à var. elem o valor na 4ª posição do lista x
- Função **len(x)** dá o número de posições/ elementos da lista x
- Listas podem ter valores de tipos diferentes: **y = [1, "A", 2, "C"]**
- Lista vazia: **vazia = [ ]**

Variáveis do tipo **list**

- Modificação das listas:
  - ***append***: adicionar elementos a listas (no final) **x.append(3)**
  - ***insert***: adicionar elemento numa dada posição **x.insert(2, "A")**
  - ***pop***: remove elemento (numa dada posição se for dada, ou no final em caso contrário) **x.pop(2)**
  - ***remove***: remove um dado elemento **x.remove("A")**
- Outras funções
  - ***sort***: ordena lista **x.sort()**
  - ***count***: conta n° ocorrências de um elemento na lista **x.count(3)**
  - ***reverse***: inverte lista **x.reverse()**

Atenção: a maior parte destas funções mudam a própria lista !!

Variáveis do tipo **list**: exemplos

```
>>> l = [2, 4, 6, 8, 10]
>>> l[3]
8
>>> l[2] = 5
>>> l
[2, 4, 5, 8, 10]
>>> l.append(12)
>>> l
[2, 4, 5, 8, 10, 12]
>>> len(l)
6
>>> l2 = [1,'a',2,'b']
>>> l2.sort()
>>> l2
[1, 2, 'a', 'b']
```

```
>>> l2.reverse()
>>> l2
['c', 3, 'a', 1]
>>> l2.insert(2,'d')
>>> l2
['c', 3, 'd', 'a', 1]
>>> l2.append(1)
>>> l2
['c', 3, 'd', 'a', 1, 1]
>>> l2.count(1)
2
>>> [0]*5
[0, 0, 0, 0, 0]
```

```
>>> l2.pop()
'b'
>>> l2
[1, 2, 'a']
>>> l2.remove(2)
>>> l2
[1, 'a']
>>> l2.extend([3,'c'])
>>> l2
[1, 'a', 3, 'c']
>>> 2 in l
True
>>> max(l)
12
```

Tuplos

- Funcionam de forma similar às *lists* mas são imutáveis (não podem ser alterados)
- Indexação, concatenação semelhantes às lists
- Funções **len**, **max**, **min**

funcionam como nas lists

- Funções que tentam

mudar tuplos dão erro

```
>>> p = (2, 5, 4)
>>> p[1]
5
>>> p[0:2]
(2, 5)
>>> p.append(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no ...
```

Strings

- Variáveis do tipo **string** – guardam sequências de caracteres.
- São definidas como sequências de caracteres entre ‘ ‘ ou “ “
- Partilham algumas funções e sintaxe com as lists
- As strings são imutáveis, i.e. o seu valor não se altera depois de serem criadas

- Em Bioinformática, estas variáveis são importantes pois permitem guardar **sequências biológicas** como DNA, RNA, proteínas.
- Sequências de DNA/RNA usam alfabeto de 4 caracteres (A,C,T/U,G);
- Sequências proteicas usam alfabeto de 21 caracteres (20 AAs + stop).

Funções pré-definidas sobre strings

- Junção de duas strings s1 e s2: **s1+s2**
- Comprimento de uma string: **len(s)**
- Caracter na posição i de uma string s: **s[i]**
- Substituir todas as ocorrências de C1 por C2 na string s

  s.replace("C1","C2")
- Converter para minúsculas ou maiusculas **s.upper() s.lower()**
- Contar o nº de vezes que um padrão p ocorre numa sequência s **s.count(p)**

Posições em Python: início da String em 0

Slicing em strings e lists

- Algumas operações de acesso a elementos de lists e strings fazem-se da mesma forma
- Intervalos de valores:
  - **s[i:j]** – todos os elementos entre as posições i e j-1
  - **s[i:]** – todos os elementos entre as posições i e o final
  - **s[:i]** – todos os elementos entre o início e a posição i-1
- Passo: pode acrescentar-se um 3° argumento que representa "saltos" na sequência
  - **s[i:j:k]** – todos os elementos entre as posições i e j-1saltando de k em k posições
  - Se passo negativo a sequência é percorrida pela ordem inversa (exemplo: **s[::-1]** inverte a sequência s)

Strings: exemplos

```
>>> seq = 'AATAGATCGA'
>>> len(seq)
10
>>> seq[5]
'A'
>>> seq[4:7]
'GAT'
>>> seq.count('A')
5
>>> seq2 = “ATAGATCTAT”
>>> seq+seq2
'AATAGATCGAATAGATCTAT’
>>> ‘1’ + “1”
‘11’
>>> seq.replace('T','U')
'AAUAGAUCGA'
```

```
>>> seq[::2]
'ATGTG'
>>> seq[::-2]
'ACAAA’
>>> seq[5:1:-2]
'AA’
>>> seq.lower()
'aatagatcga’
>>> seq.lower()[2:]
'tagatcga'
>>> seq.lower()[2:].count('c')
1
>>> c=seq.count("C")
>>> g=seq.count("G")
>>> float(c+g)/len(seq)*100
30.0
```

Procura de padrões em python: funções sobre strings/ padrões fixos

- Operador in permite verificar se um padrão fixo existe numa string
- Função find permite retornar a posição da primeira ocorrência de um padrão numa string (retorna -1 se não for encontrada)
- Função count conta nº de ocorrências de um padrão numa string
- Função replace permite substituir todas as ocorrências de um padrão por um novo padrão

Procura de padrões em python: exemplos com funções sobre strings

```
>>> "TAT" in "ATGATATATGA"
True
>>> "TATC" in "ATGATATATGA"
False
>>> str = "ATGATATATGA"
>>> "TAT" in str
True
>>> "TATC" in str
False
>>> str.find("TAT")
4
>>> str.find("TATC")
-1
>>> str.count("TA")
2
```

Dicionários

- Os dicionários guardam pares (chave, valor), sendo que não existem chaves repetidas
- Implementa conceitos de função finita (ou tabela de Hash)
- Valores de chaves e valores podem ter tipos distintos (numéricos, strings, etc)
- Criação de um dicionário:

  dict = {'A':'Ala','C':'Cys','S':'Ser'}

- Acesso a elementos: **dict['C']** ; **dict['G'] = "Glu"**

Dicionários: funções principais

- **dict.keys()** – devolve lista com as chaves
- **dict.values()** – devolve lista com os valores
- **dict.get(k)** – devolve valor associado à chave k
- **del dict[k]** – remove elemento do dicionário
- **k in dict** – verifica existência de uma chave

Dicionários: exemplos

```
>>> dic = {"Cao":"Mamifero", "Polvo":"Molusco", "Cobra":"Reptil"}
>>> dic['Cao']
'Mamifero'
>>> dic['Camarao']= 'Molusco'
>>> dic
{'Cobra': 'Reptil', 'Camarao': 'Molusco', 'Cao': 'Mamifero',
'Polvo': 'Molusco'}
>>> len(dic)
4
>>> dic.keys()
dict_keys(['Cobra', 'Camarao', 'Cao', 'Polvo']
>>> list(dic.keys())
['Cobra', 'Camarao', 'Cao', 'Polvo'
>>> "Homem" in dic
False
>>> "Cao" in dic
True
>>> del dic["Cobra"]
>>> dic
{'Camarao': 'Molusco', 'Cao': 'Mamifero', 'Polvo': 'Molusco'}
>>> list(dic.values())
['Molusco', 'Mamifero', 'Molusco']
```