1 Introdução ao estudo da matemática numérica e aritmética de máquina

Resumo: *Python é uma linguagem de programação poderosa de uso geral. Pode ser usada de forma interativa, permitindo um desenvolvimento muito rápido. Python possui muitas ferramentas de computação científica poderosas, tornando-a uma linguagem ideal para matemática aplicada e computacional. Nesta introdução apresentamos a sintaxe do Python, tipos de dados, funções e ferramentas de controle de fluxo. Esses princípios básicos do Python são uma parte essencial de quase todos os problemas que iremos resolver ao longo da disciplina.*

Vamos começar

Python tem ganho rapidamente impulso como ferramenta fundamental na computação científica.

Para iniciarmos nossa caminhada, precisamos lembrar que os arquivos Python são salvos com uma extensão `.py`. Para iniciantes, recomendamos fortemente o uso de um editor de texto simples para escrever arquivos Python, embora muitas IDEs (Ambientes de Desenvolvimento Integrados – grandes aplicativos que facilitam o desenvolvimento de código com algumas ferramentas sofisticadas) gratuitos também sejam compatíveis com Python. Por ora, quanto mais simples for o ambiente de codificação, melhor.

Um arquivo Python simples é semelhante ao código a seguir.

```
# arquivo.py
"""
    Isto eh um arquivo de cabecalho.
    O cabecalho contem informacoes basicas sobre o arquivo.
"""

if __name__ == "__main__":
    pass                            # 'pass' eh um espaco reservado temporario.
```

O caractere `#` cria um *comentário* de linha única. Os comentários são ignorados pelo intérprete e servem como anotações para o código-fonte que os acompanha. Um par de três aspas (simples ou duplas) cria uma *string* literal de múltiplas linhas, que também pode ser usada como um comentário de múltiplas linhas.

Uma *string* literal entre aspas triplas no topo do arquivo serve como *cabeçalho* para o arquivo. O cabeçalho normalmente identifica o autor e inclui instruções sobre como usar o arquivo. O código Python executável vem depois do cabeçalho.

Problema 1. Abra um arquivo com o nome `python_intro.py` (ou crie o arquivo no editor de texto, caso ainda não tenha o arquivo editado).

Adicione as seguintes informações ao cabeçalho na parte superior.

```
if __name__ == "__main__":
    print("Ola mundo!")  # Identacao com quatro espacos (Nao foi usado tab↩
        ).
```

Tenha certeza de que o arquivo foi salvo. Abra um terminal de *prompt* (*Terminal* em Linux ou Mac e *Command Prompt* ou *GitBash* em Windows) e navegue até o diretório onde o arquivo foi salvo. Use o comando `ls` (or `DIR` em Windows) para listar os arquivos e pastas do diretório corrente, `pwd` (`cd ,` em Windows) para imprimir o diretório em que está trabalhando, e `cd` para mudar entre os diretórios.

```
$ pwd                              # Print the working directory.
/Users/Guest
$ ls                               # List the files and folders here.
Desktop     Documents       Downloads       Pictures        Music
$ cd Documents                     # Navigate to a different folder.
$ pwd
/Users/Guest/Documents
$ ls                               # Check to see that the file is here.
python_intro.py
```

Tendo encontrado o arquivo Python, ele pode ser executado através do comando:

```
$ python python_intro.py
```

Se `Ola Mundo!` for exibido na sua tela, a execução do programa ocorreu como esperado!

Nota

A expressão `if __name__ == "__main__"` é extremamente útil para criar funções de teste e depurar seu código. Para usar `if __name__ == "__main__"` para testar funções declaradas, **deve ser colocado no final do seu arquivo** com o código que você deseja executar diretamente a seguir. Se você tentar executar esse bloco de código no início do arquivo com funções que são declaradas posteriormente, você receberá um erro de ter funções indefinidas.

IPython

Python pode rodar de maneira interativa usando alguma interfaces. A mais básicas dessas ferramentas é o *IPython*[1] - Interpretador Python. No IPython os triplos colchetes angulares indicam que o código fornecido será executado na linha pelo Interpretador Python.

Para executar um *script* no IPython, use o comando `%run`.

```
$ ipython                           # Start IPython.

In [1]: print("Isso eh simplesmente IPython!")   # Execute.
Isso eh simplesmente IPython!

In [2]: %run python_intro.py       # Rodar uma script particular Python.
Ola mundo!
```

Python é uma linguagem de programação **orientada a objetos**. Podemos relembrar das características desse tipo de programação:

- Um **objeto** refere-se a uma instância específica de uma classe.
- **Atributos** são as características ou propriedades de um objeto que armazena dados.
- **Métodos** definem as ações ou comportamentos que um objeto pode executar.

Uma das maiores vantagens do IPython é que ele suporta *introspecção de objetos*, enquanto o interpretador Python regular não. A introspecção de objetos revela rapidamente todos os métodos e atributos associados a um objeto. O IPython também possui uma função `help()` integrada que fornece ajuda interativa.

```
# Uma lista eh uma estrutura de dados basica do Python. Para ver os metodos ↩
    associados a
# uma lista, digite o nome-objeto (list), seguido por um ponto final e ↩
    pressione tab.
In [1]: list.   # Pressione 'tab'.
              append()  count()   insert()  remove()
              clear()   extend()  mro()     reverse()
              copy()    index()   pop()     sort()

# Para saber mais sobre um metodo especifico, use um '?' e pressione 'Enter'.
In [1]: list.append?
Signature: list.append(self, object, /)
Docstring: Append object to the end of the list.
Type:      method_descriptor

In [2]: help()                      # Start IPython's interactive help utility.

help> list                          # Get documentation on the list class.
Help on class list in module builtins:
```

[1] Acesse o endereço: `https://ipython.org/`

```
class list(object)
 |  list(iterable=(),/)
 |  # ...                              # Press 'q' to exit the info screen.

help> quit                             # End the interactive help session.
```

> **Nota**
>
> Use o IPython lado a lado com um editor de texto para testar rapidamente a sintaxe e pequenos trechos de código. Testar pequenos pedaços de código no IPython **antes** de colocá-los em um programa revela erros e acelera bastante o processo de codificação. Consulte a internet com dúvidas; `stackoverflow.com` é um recurso particularmente valioso para responder perguntas comuns de programação.
>
> A melhor maneira de aprender uma nova linguagem de codificação é escrevendo código. Acompanhe os exemplos nas caixas de código amarelas deste curso, executando-os em um console IPython. Evite copiar e colar por enquanto; seus dedos também precisam aprender o idioma.

Python Básico

Aritmética

Python pode ser usado como uma calculadora, com os operadores básicos regulares: `+`, `-`, `*`, and `/`. Use `**` para exponenciação e `%` para divisão modular.

```
>>> 3**2 + 2*5                         # Python obedece ordem de operacoes.
19

>>> 13 % 3                             # O operador % calcula o
1                                      # resto da divisao: 13 = (3*4) + 1.
```

Em muitos interpretadores Python, o caracter *underscore* (sublinhado) `_` é uma variável com o valor da saída do comando anterior, como o botão ANS em muitas calculadoras.

```
>>> 12 * 3
36
>>> _ / 4
9.0
```

Os colchetes angulares, de comparação de dados, `<` e `>`, atuam como esperado. O operador `==` verifica a igualdade numérica e os operadores `<=` e `>=` correspondem às desigualdades $\leq$ (menor ou igual) e $\geq$ (maior ou igual), respectivamente. Para conectar multiplas expressões booleanas utilize os operadores `and`, `or` e `not`[2].

[2] Em muitas outras linguagens de programação os operadores `and`, `or`, e `not` são escritos como `&&`, `||` e `!`, respectivamente. A convenção de Python é muito mais objetiva não requer a adição de marcadores como parênteses, por exemplo.

```
>>> 3 > 2.99
True
>>> 1.0 <= 1 or 2 > 3
True
>>> 7 == 7 and not 4 < 4
True

>>> True and True and True and True and True and False
False
>>> False or False or False or False or False or True
True
>>> True or not True
True
```

Variaveis

Um sinald e igualdade **único** = atribui um ou mais valores (à direita) a um ou mais nomes de variáveis (à esquerda). Um **duplo** sinal de igualdade == é um operador de comparação que retorna `True` ou `False`, como no bloco de código anterior.

Ao contrário de muitas linguagens de programação, Python não exige que o tipo de dados de uma variável seja especificado na inicialização. Por causa disso, Python é chamado de linguagem *digitada dinamicamente*.

```
>>> x = 12                          # Initialize x with the integer 12.
>>> y = 2 * 6                       # Initialize y with the integer 2*6 = 12.
>>> x == y                          # Compare the two variable values.
True

>>> x, y = 2, 4                     # Give both x and y new values in one line.
>>> x == y
False
```

Funções

Para definir uma função, use a palavra-chave `def` seguida pelo nome da função, uma lista de parâmetros entre parênteses e dois pontos. Em seguida, recue o corpo da função usando exatamente **quatro** espaços.

```
>>> def add(x, y):
...     return x + y                # Indent with four spaces.
```

Atenção!

Muitas outras linguagens usam chaves `{}` para delimitar blocos, mas Python usa recuo de espa-

ços em branco. Na verdade, espaços em branco são essencialmente a única coisa que o Python é particularmente exigente em comparação com outras linguagens: **misturar tabulações e espaços confunde o interpretador e causa problemas**. A maioria dos editores de texto tem uma configuração para definir o tipo de recuo para espaços, para que você possa usar a tecla tab do teclado para inserir quatro espaços (às vezes chamados de *soft tabs*). Para consistência, **nunca** use tabulações; **sempre** use espaços.

Funções são definidas com *parâmetros* e chamadas com *argumentos*, embora os termos sejam frequentemente usados de forma intercambiável. Abaixo, `width` e `height` são parâmetros para a função `area()`. Os vares `2` e `5` são os argumentos passados ao chamar a função.

```
>>> def area(width, height):        # Define the function.
...     return width * height
...
>>> area(2, 5)                      # Call the function.
10
```

As funções Python também podem retornar vários valores.

```
>>> def arithmetic(a, b):
...     return a - b, a * b         # Separate return values with commas.
...
>>> x, y = arithmetic(5, 2)         # Unpack the returns into two variables.
>>> print(x, y)
3 10
```

A palavra-chave `lambda` é um atalho para criar funções de uma linha. Por exemplo, os polinômios $f(x) = 6x^3 + 4x^2 - x + 3$ e $g(x, y, z) = x + y^2 - z^3$ podem ser definidos como funções em uma linha cada.

```
# Define the polynomials the usual way using 'def'.
>>> def f(x):
...     return 6*x**3 + 4*x**2 - x + 3
>>> def g(x, y, z):
...     return x + y**2 - z**3

# Equivalently, define the polynomials quickly using 'lambda'.
>>> f = lambda x: 6*x**3 + 4*x**2 - x + 3
>>> g = lambda x, y, z: x + y**2 - z**3
```

Nota

A documentação é importante em todas as linguagens de programação. Cada função deve ter um *docstring* — uma *string* literal entre aspas triplas, logo abaixo da declaração da função — que descreve o propósito da função, as entradas esperadas e valores de retorno, bem como quaisquer outras notas que sejam importantes para o usuário. Documentos curtos são aceitáveis para funções muito simples, porém funções mais complicadas requerem explicações cuidadoas

e detalhadas.

```
>>> def add(x, y):
...     """Return the sum of the two inputs."""
...     return x + y

>>> def area(width, height):
...     """Return the area of the rectangle with the specified width
...     and height.
...     """
...     return width * height
...
>>> def arithmetic(a, b):
...     """Return the difference and the product of the two inputs."""
...     return a - b, a * b
```

As funções Lambda não podem ter docstrings personalizados, portanto a palavra-chave `lambda` deve ser usada apenas como um atalho para funções muito simples ou intuitivas que não precisam de rotulagem adicional.

Problema 2. O volume de uma esfera com raio r é $V = \frac{4}{3}\pi r^3$. Em seu arquivo Python do problema 1, defina uma função chamada `sphere_volume()` que aceita um único parâmetro r. Retorne o volume da esfera de raio r, usando $3,14159$ como aproximação para π (por enquanto). Escreva também uma documentação apropriada para sua função.

Para testar sua função, chame-a na cláusula `if __name__ == "__main__"` e imprima o valor retornado. Execute seu arquivo para ver se sua resposta é o que você espera.

Atenção!

A instrução `return` encerra instantaneamente a chamada da função e passa o valor de retorno para o chamador da função. No entanto, as funções não precisam ter uma instrução return. Uma função sem uma instrução return retorna implicitamente a constante Python `None`, que é semelhante ao valor especial `null` de muitas outras linguagens. Chamar `print()` no final de uma função **não** faz com que a função retorne quaisquer valores.

```
>>> def oops(i):
...     """Increment i (but forget to return anything)."""
...     print(i + 1)
...
>>> def increment(i):
...     """Increment i."""
...     return i + 1
...
```

```
>>> x = oops(1999)                    # x contains 'None' since oops()
2000                                  # doesn't have a return statement.
>>> y = increment(1999)               # However, y contains a value.
>>> print(x, y)
None 2000
```

Se você tiver alguma intenção de usar os resultados de uma função, use uma instrução `return`.

Também é possível especificar *valores padrão* para os parâmetros de uma função. No exemplo a seguir, a função `pad()` possui três parâmetros e o valor de `c` é padronizado como 0. Se não for especificado na chamada da função, a variável `c` conterá o valor 0 quando a função for executada.

```
>>> def pad(a, b, c=0):
...     """Print the arguments, plus a zero if c is not specified."""
...     print(a, b, c)
...
>>> pad(1, 2, 3)                  # Specify each parameter.
1 2 3
>>> pad(1, 2)                     # Specify only non-default parameters.
1 2 0
```

É importante observar que os argumentos posicionais devem preceder os argumentos nomeados em uma chamada de função. Além disso, os parâmetro sem valores padrão devem preceder os parâmetros com valores padrão em uma definição de função. Por exemplo, `a` e `b` devem vir antes de `c` na definição da função `pad()`. Examine os blocos de código a seguir que demonstram como argumentos posicionais e nomeados são usados para chamar uma função.

```
# Try defining pad() with a named argument before a positional argument.
>>> def pad(c=0, a, b):
...     print(a, b, c)
...
SyntaxError: non-default argument follows default argument
```

```
# Correctly define pad() with the named argument after positional arguments.
>>> def pad(a, b, c=0):
...     """Print the arguments, plus a zero if c is not specified."""
...     print(a, b, c)
...

# Call pad() with 3 positional arguments.
>>> pad(2, 4, 6)
2 4 6

# Call pad() with 3 named arguments. Note the change in order.
>>> pad(b=3, c=5, a=7)
7 3 5
```

```
# Call pad() with 2 named arguments, excluding c.
>>> pad(b=1, a=2)
2 1 0

# Call pad() with 1 positional argument and 2 named arguments.
>>> pad(1, c=2, b=3)
1 3 2
```

Problema 3. A função integrada `print()` possui os argumentos de palavras-chave úteis `sep` e `end`. Ele aceita qualquer número de argumentos posicionais e os imprime com `sep` inserido entre valores (padrão para um espaço), então imprime `end` (padrão para o *caractere de nova linha* `'\n'`).

Escreva uma função chamada `isolate()` que aceite cinco argumentos. A função deve imprimir os três primeiros argumentos separados por 5 espaços e depois imprimir os dois últimos argumentos com um único espaço separando os três últimos argumentos. Por exemplo,

```
>>> isolate(1, 2, 3, 4, 5)
1     2     3 4 5
```

ATENÇÃO!

Nas versões anteriores do Python, `print()` era uma *instrução* (como `return`), não uma função, e poderia, portanto, ser executada sem parênteses. No entanto, faltavam argumentos de palavras-chave como `sep` e `end`. Se você estiver usando Python 2.7, inclua a seguinte linha no topo do arquivo para transformar a instrução `print` na nova função `print()`.

```
>>> from __future__ import print_function
```

Tipos e Estruturas de Dados

Tipos Numéricos

Python tem quatro tipos de dados numéricos: `int`, `long`, `float`, and `complex`. Cada um deles armazena um tipo diferente de número. Each stores a different kind of number.

A função integrada `type()` identifica o tipo de dados de um objeto.

```
>>> type(3)                          # Numbers without periods are integers.
int

>>> type(3.0)                        # Floats have periods (3. is also a float).
float
```

Python possui dois tipos de divisão: inteiro e flutuante. O operador `/` realiza a divisão flutuante (divisão fracionária verdadeira), e o operador `//` realiza a divisão inteira, que arredonda o resultado para o próximo número inteiro. Se ambos os operandos de `//` forem números inteiros, o resultado será um `int`. Se um ou ambos os operandos forem flutuantes, o resultado será um `float`. A divisão regular com `/` sempre retorna um `float`.

```
>>> 15 / 4                          # Float division performs as expected.
3.75
>>> 15 // 4                         # Integer division rounds the result down.
3
>>> 15. // 4
3.0
```

Atenção!

Nas versões anteriores do Python, usar `/` com dois inteiros realizava a divisão de inteiros, mesmo nos casos em que a divisão não era par. Isso pode resultar em alguns erros incrivelmente sutis e frustrantes. Se você estiver usando Python 2.7, sempre inclua um `.` nos operandos ou converta pelo menos um como float quando desejar divisão float.

```
# PYTHON 2.7
>>> 15 / 4                            # The answer should be 3.75, but the
3                                     # interpreter does integer division!

>>> 15. / float(4)                    # 15. and float(4) are both floats, so
3.75                                  # the interpreter does float division.
```

Como alternativa, incluir a linha a seguir na parte superior do arquivo redefine os operadores `/` e `//` para que sejam tratados da mesma forma que no Python 3.

```
>>> from __future__ import division
```

Python também oferece suporte a cálculos de números complexos, emparelhando dois números como partes reais e imaginárias. Use a letra j, e não i, para a parte imaginária.

```
>>> x = complex(2, 3)                 # Create a complex number this way...
>>> y = 4 + 5j                        # ...or this way, using j (not i).
>>> x.real                            # Access the real part of x.
2.0
>>> y.imag                            # Access the imaginary part of y.
5.0
```

Strings

Em Python, *strings* são criadas com aspas simples ou duplas. Para concatenar duas ou mais *strings*, use o operador `+` entre variáveis de *strings* ou literais.

```
>>> str1 = "Ola"
>>> str2 = 'mundo'
>>> my_string = str1 + " " + str2 + '!'
>>> my_string
'Hello world!'
```

Partes de uma *string* podem ser acessadas usando *slicing*, indicado por colchetes `[ ]`. A sintaxe de fatiamento é `[start:stop:step]`. Os parâmetros `start` e `stop` são padronizados para o início e o fim da string, respectivamente. O parâmetro `step` tem como padrão 1.

```
>>> my_string = "Hello world!"
>>> my_string[4]                # Indexing begins at 0.
'o'
>>> my_string[-1]               # Negative indices count backward from the end.
'!'

# Slice from the 0th to the 5th character (not including the 5th character).
>>> my_string[:5]
'Hello'

# Slice from the 6th character to the end.
>>> my_string[6:]
'world!'

# Slice from the 3rd to the 8th character (not including the 8th character).
>>> my_string[3:8]
'lo wo'

# Get every other character in the string.
>>> my_string[::2]
'Hlowrd'
```

Problema 4. Escreva duas novas funções, chamadas `first_half()` e `backward()`.

1. `first_half()` deve aceitar um parâmetro e retornar a primeira metade dele, excluindo o caractere do meio se houver um número ímpar de caracteres.
   (Hint: the built-in function `len()` returns the length of the input.)

2. The `backward()` function should accept a parameter and reverse the order of its characters using slicing, then return the reversed parameter.
   (Dica: o parâmetro `step` usado no fatiamento pode ser negativo.)

Use o IPython para testar rapidamente a sintaxe de cada função.

Listas

Uma lista (`list`) em Python é criada quando utilizamos colchetes (quadrados, `[ ]`) e separamos seus valores por vírgulas. Os elementos de uma lista não precisar ser do mesmo tipo. O acesso às entradas em uma lista ocorrem por meio de operações de indexação ou fatiamento.

```
>>> my_list = ["Ola", 93.8, "mundo", 10]
>>> my_list[0]
'Ola'
>>> my_list[-2]
'mundo'
>>> my_list[:2]
['Ola', 93.8]
```

As listas aceitam os seguintes métodos: `append()`, `insert()`, `remove()`, `extend()`, e `pop()`.

```
>>> my_list = [1, 2]                # Create a simple list of two integers.
>>> my_list.append(4)               # Append the integer 4 to the end.
>>> my_list.insert(2, 3)            # Insert 3 at location 2.
>>> my_list
[1, 2, 3, 4]
>>> my_list.remove(3)               # Remove 3 from the list.
>>> my_list.pop()                   # Remove (and return) the last entry.
4
>>> my_list
[1, 2]
>>> my_list.extend([5, 6])           # Append multiple values at once.
>>> my_list
[1, 2, 5, 6]
```

O fatiamento também é muito útil para substituir valores em uma lista.

```
>>> my_list = [10, 20, 30, 40, 50]
>>> my_list[0] = -1
>>> my_list[3:] = [8, 9]
>>> print(my_list)
[-1, 20, 30, 8, 9]
```

O operador `in` verifica rapidamente se um determinado valor está em uma lista (ou outro iterável, incluindo strings).

```
>>> my_list = [1, 2, 3, 4, 5]
>>> 2 in my_list
True
>>> 6 in my_list
```

```
False
>>> 'a' in "xylophone"              # 'in' also works on strings.
False
```

Exercícios

1. Converter as seguintes expressões para o interpretador Python:

   a) $10 + 20 \times 30$

   b) $4^2 \div 30$

   c) $(9^4 + 2) \times 6 - 1$

2. No interpretador `Python` (ou num `arquivo.py`) resolva a expressão: $10\%3 \times 10^2 + 1 - 10 \times 4/2$. Resolva também usando apenas lápis e papel. O resultado a ser encontrado é 81.0

3. Faça um programa que resolva a soma de três variáveis e imprima o resultado na tela.

4. Considerando um salário de R$ 750.00, determine numericamente um aumento de 15%.

5. Escreva um programa que, a partir da entradas dos valores de dias, horas, minutos e segudos, calcule o tempo total em segundos.

6. Escreva um programa que pergunte três números para o usuário e que devolva como resposta qual o maior deles.

7. Escreva um programa em Python que verifique se um número é primo.