Introdução à Física Computacional I (4300218)

Prof. André Vieira
apvieira@if.usp.br
Sala 3120 – Edifício Principal

Aula 1

Programação em Python para físicos: programação básica

Por que Python?

- Facilidade para aprender
- Simplicidade na utilização
- Disponibilidade de recursos poderosos

Um exemplo

Um código simples em Python

```
# eg1-names.py: output three names to the console.

names = ['Isaac Newton', 'Marie Curie', 'Albert Einstein']
for name in names:
    print(name)
```

Resultado:

```
Isaac Newton
Marie Curie
Albert Einstein
```

Um exemplo

Para produzir o mesmo resultado em C++:

```
/* eg1-names.c: output three names to the console. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STRING_LENGTH 20
#define NUMBER_OF_STRINGS 3

int main()
{
int i;
char names[NUMBER_OF_STRINGS][MAX_STRING_LENGTH+1];
strcpy(names[0], "Isaac Newton");
strcpy(names[1], "Marie Curie");
strcpy(names[2], "Albert Einstein");

for (i=0;i<NUMBER_OF_STRINGS;i++) {
    fprintf(stdout, "%s\n", names[i]);
}

return EXIT_SUCCESS;
}
```

Por que (não) Python?

- Facilidade para aprender
- Simplicidade na utilização
- Disponibilidade de recursos poderosos
- Velocidade de execução

Ferramentas

- Iniciem os computadores no ‘linux’
- Localizem o ambiente de desenvolvimento ‘IDLE’
- Na *shell* que for aberta, cliquem em ‘File → New File’
- Na janela de edição que for aberta (não na shell!), cliquem em ‘Save as’ e escolham um nome para o arquivo que termine em ‘.py’, tal como ‘teste.py’.

Variáveis e atribuições

- Uma variável armazena um valor (numérico ou não) que pode ser atribuído através de uma instrução. Exemplos disso são:

  `x = 1` variável do tipo inteiro (*int*)

  `y = 2.5` variável do tipo real (*float*)

  `z = 1+2j` variável do tipo complexo (*complex*)

  `a = "chave"` variável do tipo texto (*string*)

- Os espaços nas atribuições não são obrigatórios, mas incluí-los deixa o código mais claro de ler.

Variáveis e atribuições

- O Python reconhece o tipo de variável pela atribuição, e esse tipo pode mudar ao longo de um programa (um conjunto de instruções):

```
x = 1
```
`x` é variável do tipo inteiro (*int*)

⋮

```
x = 2.5
```
`x` agora é do tipo real (*float*)

⋮

```
x = "chave"
```
`x` agora é do tipo texto (*string*)

- Note que, embora permitida, essa não é uma prática recomendável.

Variáveis e atribuições

- Como o tipo da variável é definido na atribuição, se você desejar que uma variável possa assumir valores reais mas tenha inicialmente um valor inteiro (por exemplo 2), use uma das formas a seguir:

```
x = 2.0
```

ou

```
x = float(2)
```

Variáveis e atribuições

- Analogamente, se você desejar que uma variável possa assumir valores complexos mas tenha inicialmente um valor real (por exemplo 4.25), use uma das formas a seguir:

```
x = 4.25 + 0j
```

ou

```
x = complex(4.25)
```

Instruções de entrada e saída

- Para visualizar na shell o valor de uma ou mais variáveis, ou ainda alguma mensagem, utilize a função `print`.
- Exemplo 1:

```
x = 1

print(x)

x = 1.5

print(x)
```

```
>>>
========== RESTART: /private/tmp/temp.py ==========
1
1.5
>>>
```

Instruções de entrada e saída

- Para visualizar na shell o valor de uma ou mais variáveis, ou ainda alguma mensagem, utilize a função `print`.
- Exemplo 2:

```
x = 1

y = 2

print(x,y)
```

```
========== RESTART: /private/tmp/temp.py ==========
1 2
>>>
```

Instruções de entrada e saída

- Para visualizar na shell o valor de uma ou mais variáveis, ou ainda alguma mensagem, utilize a função `print`.
- Exemplo 3:

```
x = 1

y = 2

print("O valor de x é",x,"e o de y é",y)
```

```
========== RESTART: /private/tmp/temp.py ==========
O valor de x é 1 e o de y é 2
>>>
```

Instruções de entrada e saída

- Para visualizar na shell o valor de uma ou mais variáveis, ou ainda alguma mensagem, utilize a função `print`.
- Exemplo 4:

```
x = 2.5

z = 3+4j

print(x,z,sep="...")
```

```
========== RESTART: /private/tmp/temp.py ==========
2.5...(3+4j)
>>>
```

Instruções de entrada e saída

- Para solicitar a digitação de uma variável durante a execução de um programa utilize a instrução `input`.
- Exemplo 1:

```
x = input("Entre com o valor de x: ")
print("O valor de x é",x)
```

```
========== RESTART: /private/tmp/temp.py ==========
Entre com o valor de x: 3.14
O valor de x é 3.14
>>>
========== RESTART: /private/tmp/temp.py ==========
Entre com o valor de x: Olá
O valor de x é Olá
>>>
```

Instruções de entrada e saída

- Exemplo 2: forçando a variável a ser real

```
y = input("Entre com o valor de x: ")
x = float(y)
print("O valor de x é",x)
```

```
========== RESTART: /private/tmp/temp.py ==========
Entre com o valor de x: 2
O valor de x é 2.0
>>>
========== RESTART: /private/tmp/temp.py ==========
Entre com o valor de x: Olá
Traceback (most recent call last):
  File "/private/tmp/temp.py", line 2, in <module>
    x = float(y)
ValueError: could not convert string to float: 'Olá'
>>>
```

- Analogamente para `int(y)` e `complex(y)`

Aritmética

- As operações matemáticas básicas são escritas em Python da seguinte forma:

`x + y`	adição
`x - y`	subtração
`x * y`	multiplicação
`x / y`	divisão
`x ** y`	elevar `x` à potência `y`
`x // y`	divisão inteira
`x % y`	resto da divisão de `x` por `y`

Aritmética

- Exemplo 1:

```
========== RESTART: /private/tmp/temp.py ==========
O valor de x é 2 e o de y é 3
O resultado de x + y é 5
O resultado de x - y é -1
O resultado de x * y é 6
O resultado de x / y é 0.6666666666666666
O resultado de x ** y é 8
O resultado de x // y é 0
O resultado de x % y é 2
>>>
```

- Exemplo 2:

```
========== RESTART: /private/tmp/temp.py ==========
O valor de x é 2.0 e o de y é 3.0
O resultado de x + y é 5.0
O resultado de x - y é -1.0
O resultado de x * y é 6.0
O resultado de x / y é 0.6666666666666666
O resultado de x ** y é 8.0
O resultado de x // y é 0.0
O resultado de x % y é 2.0
>>>
```

Aritmética

- Exemplo 3:

```
========== RESTART: /private/tmp/temp.py ==========
O valor de x é (2+0j) e o de y é (3+0j)
O resultado de x + y é (5+0j)
O resultado de x - y é (-1+0j)
O resultado de x * y é (6+0j)
O resultado de x / y é (0.6666666666666666+0j)
O resultado de x ** y é (8+0j)
Traceback (most recent call last):
  File "/private/tmp/temp.py", line 9, in <module>
    print("O resultado de x // y é",x // y)
TypeError: can't take floor of complex number.
>>>
```

- Exemplo 4:

```
========== RESTART: /private/tmp/temp.py ==========
O valor de x é dois e o de y é três
O resultado de x + y é doistrês
Traceback (most recent call last):
  File "/private/tmp/temp.py", line 5, in <module>
    print("O resultado de x - y é",x - y)
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>>
```

Aritmética

- Exemplo 5:

```
========== RESTART: /private/tmp/temp.py ==========
O valor de x é 2 e o de y é 3.0
O resultado de x + y é 5.0
O resultado de x - y é -1.0
O resultado de x * y é 6.0
O resultado de x / y é 0.6666666666666666
O resultado de x ** y é 8.0
O resultado de x // y é 0.0
O resultado de x % y é 2.0
>>>
```

- Exemplo 6:

```
========== RESTART: /private/tmp/temp.py ==========
O valor de x é 2 e o de y é (3+0j)
O resultado de x + y é (5+0j)
O resultado de x - y é (-1+0j)
O resultado de x * y é (6+0j)
O resultado de x / y é (0.6666666666666666+0j)
O resultado de x ** y é (8+0j)
Traceback (most recent call last):
  File "/private/tmp/temp.py", line 9, in <module>
    print("O resultado de x // y é",x // y)
TypeError: can't take floor of complex number.
>>>
```

Aritmética: regras de precedência

- Como em expressões “analógicas”, a convenção é de que multiplicações e divisões têm precedência sobre adição e subtração. Potenciação tem precedência sobre tudo.

```
========== RESTART: /private/tmp/temp.py ==========
O valor de 2 + 3 * 4 é 14 e não 20
O valor de 2 + 3 / 4 é 2.75 e não 1.25
O valor de 2 - 3 ** 4 é -79 e não 1
O valor de 2 * 3 ** 4 é 162 e não 1296
>>>
```

- Operações de mesma precedência são realizadas da esquerda para a direita, exceto a potenciação, que é executada da direita para a esquerda.

```
========== RESTART: /private/tmp/temp.py ==========
O valor de 2 * 3 // 4 é 1 e não 0
O valor de 2 ** 3 ** 2 é 512 e não 64
>>>
```

Aritmética: regras de precedência

- Para contornar as regras de precedência, basta utilizar parênteses:

```
x = a + b/c

x = (a + b)/c

x = a + 2*b - 0.5*(1.618**c + 2/7)
```

Aritmética e atribuições

- O sinal de igualdade nas atribuições em Python **não** tem o mesmo significado que em equações:

  `2*x = y` não é uma instrução válida

  `x = x + 1` é uma instrução válida

  `x = x**2 + 2*x` é uma instrução válida

- Se o valor inicial de `x` em cada caso é `3`, quais são os valores de `x` produzidos por cada instrução válida acima?

Aritmética e atribuições

- O sinal de igualdade nas atribuições em Python **não** tem o mesmo significado que em equações:

  `2*x = y` não é uma instrução válida;

  `x = x + 1` é uma instrução válida;

  `x = x**2 + 2*x` é uma instrução válida.

- Se o valor inicial de `x` em cada caso é `3`, quais são os novos valores de `x` produzidos por cada instrução válida acima?

  `x=4` e `x=15`

Aritmética e atribuições

- Algumas atribuições podem ser escritas de forma compacta com o auxílio de *modificadores*:

  `x += 1` equivale a `x = x + 1`

  `x -= 2` equivale a `x = x - 2`

  `x *= 1.5` equivale a `x = 1.5*x`

  `x /= 3.14` equivale a `x = x/3.14`

- Python permite múltiplas atribuições por linha:

```
x,y = 2,3
x,y = y,x
```

Quantas linhas são necessárias para fazer as mesmas atribuições individualmente?

Exercício 1

Escreva um programa que defina uma variável `a` igual ao número real 2, outra variável `b` igual à unidade imaginária i, uma terceira variável `c` igual à soma das duas primeiras e, finalmente, imprima o valor dessa última variável.

Exercício 1: solução

Escreva um programa que defina uma variável `a` igual ao número real 2, outra variável `b` igual à unidade imaginária i, uma terceira variável `c` igual à soma das duas primeiras e, finalmente, imprima o valor dessa última variável.

temp.py - /private/tmp/temp.py (3.7.4)

```
a,b = 2,1j
c = a + b
print("O valor de c, igual a a+b, é",c)
print("A parte real de c vale",c.real)
print("A parte imaginária de c vale",c.imag)
```

```
========== RESTART: /private/tmp/temp.py ==========
O valor de c, igual a a+b, é (2+1j)
A parte real de c vale 2.0
A parte imaginária de c vale 1.0
>>>
```

Note a utilização de `.real` e `.imag` para o acesso a *atributos* do *objeto* `c`.

Exercício 2

Uma bola é abandonada do repouso do alto de uma torre. Escreva um programa que peça ao usuário para digitar a altura da torre em metros e então calcule o tempo que a bola leva para atingir o solo, ignorando a resistência do ar. Utilize o programa para calcular esse tempo para uma altura de 100 metros.

Exercício 2: solução

Uma bola é abandonada do repouso do alto de uma torre. Escreva um programa que peça ao usuário para digitar a altura da torre em metros e então calcule o tempo que a bola leva para atingir o solo, ignorando a resistência do ar. Utilize o programa para calcular esse tempo para uma altura de 100 metros.

temp.py - /private/tmp/temp.py (3.7.4)

```
# Vamos primeiro pedir que o usuário digite a altura inicial
h = float(input("Digite a altura inicial: "))
# A aceleração da gravidade local é uma constante importante
g = 9.81 # Implicitamente usamos unidades do SI
# Agora o cálculo da resposta, que vem de 0 = h - (1/2)*g*t**2
t = (2*h/g)**(1/2)
print("O tempo de queda é",t)
```

```
========== RESTART: /private/tmp/temp.py ==========
Digite a altura inicial: 100
O tempo de queda é 4.515236409857309
>>>
```

Note a utilização de # para iniciar um *comentário*.

Funções, pacotes e módulos

- Python tem algumas funções internas, que já utilizamos, como `float` e `print`. Entre as demais funções internas, talvez a mais útil em física seja a função `abs`.

  `abs(-3)` retorna `3`

  `abs(-4.0)` retorna `4.0`

  `abs(3-4j)` retorna `5.0`

- Uma lista completa das funções internas (*built-in functions*) do Python está em https://docs.python.org/3/library/functions.html

Funções, pacotes e módulos

- Para operações mais complicadas, como cálculos envolvendo funções trascendentais, matrizes e gráficos, por exemplo, é preciso invocar *pacotes*.
- Um pacote essencial em física é o `math`, que dá acesso a muitas funções e constantes matemáticas, tais como `log`, `sqrt`, `cos`, `pi`. Uma lista completa está em https://docs.python.org/3/library/math.html
- Para usar uma função do pacote, deve-se invocá-la:

```
from math import log

x = log(2.5)
```

Funções, pacotes e módulos

- Você pode invocar mais de uma função ou constante na mesma linha de código:

```
from math import log,pi,e

x = log(pi*e)
```

- Se quiser invocar todas as funções e constantes:

```
from math import *
```

- Alguns pacotes possuem “subpacotes”, ou *módulos*, que podem ser invocados assim:

```
from numpy.linalg import inv
```

Exercício 3

Crie um programa com as linhas abaixo e o execute. O que você pode concluir a partir dos resultados?

```
from math import cos,pi

print(cos(pi),cos(180))

d,e = 8,2

from math import *

print(sqrt(d ** e))
```

Exercício 3: resultado

Crie um programa com as linhas abaixo e o execute. O que você pode concluir a partir dos resultados?

```
from math import cos,pi

print(cos(pi),cos(180))

d,e = 8,2

from math import *

print(sqrt(d ** e))
```

```
========== RESTART: /private/tmp/temp.py ==========
-1.0 -0.5984600690578581
16.88210319127114
>>>
```

As funções trigonométricas esperam argumentos em radianos.
Importar constantes (e funções) de um pacote subscreve os valores definidos anteriormente.

Exercício 4

Segundo a relatividade especial, se Δt_1 é o intervalo de tempo entre dois eventos segundo um observador inercial, o intervalo de tempo correspondente Δt_2 para um segundo observador que se move com velocidade v em relação ao primeiro é igual a

$$\Delta t_2 = \Delta t_1 \sqrt{1-(v/c)^2}.$$

sendo c a velocidade da luz. Uma espaçonave viaja da Terra em linha reta a uma velocidade v até um outro planeta a uma distância de x anos-luz. Escreva um programa que solicite ao usuário o valor de x e a velocidade v como fração da velocidade da luz, e em seguida imprima o tempo em anos que a espaçonave leva para completar a viagem (a) no referencial de repouso de um observador na Terra e (b) como percebido por um passageiro a bordo da nave. Use seu programa para calcular respostas para um planeta a 10 anos-luz e uma velocidade $v = 0.99c$.

Exercício 4: solução

exercicio4.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula01/exercicio4.p...

File Edit Format Run Options Window Help

```
from math import sqrt
x = float(input("Digite a distância em anos-luz: "))
v = float(input("Digite a velocidade como fração de c: "))
# Intervalo de tempo (em anos) para um observador na Terra
delta_t1 = x/v
# Intervalo de tempo (em anos) para um passageiro da nave
delta_t2 = delta_t1 * sqrt(1-v**2)
print("Tempo de viagem para um observador na Terra:",delta_t1)
print("Tempo de viagem para um passageiro da nave:",delta_t2)
```

```
 RESTART: /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula01/exercicio4.py
Digite a distância em anos-luz: 10
Digite a velocidade como fração de c: 0.99
Tempo de viagem para um observador na Terra: 10.1010101010101
Tempo de viagem para um passageiro da nave: 1.424922826228878
>>>
```

Para a próxima aula

- Exercícios no moodle (não são ainda um dos trabalhos utilizados para a nota no curso)