MAP 2112 – Introdução à Lógica de Programação e Modelagem Computacional

1º Semestre - 2023

Prof. Dr. Luis Carlos de Castro Santos

lsantos@ime.usp.br

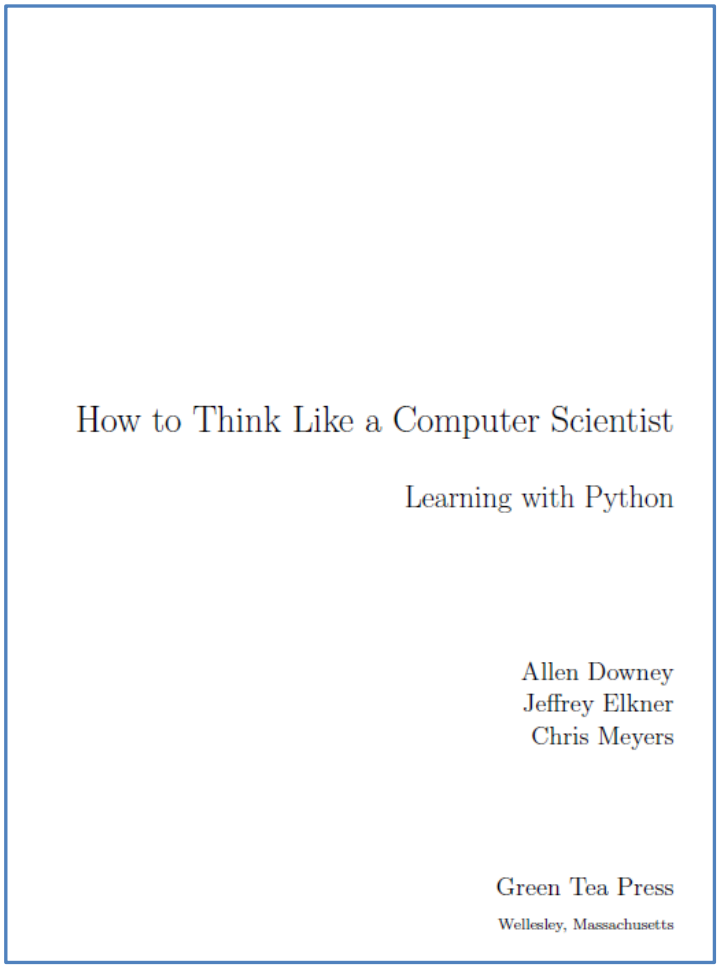
AJUSTE DE CURSO – Roteiro da Disciplina

Ao invés de adaptar o curso de R para Python seguirei o roteiro no curso de Python introdutório de Ciência da Computação.

Seguirei o material original:

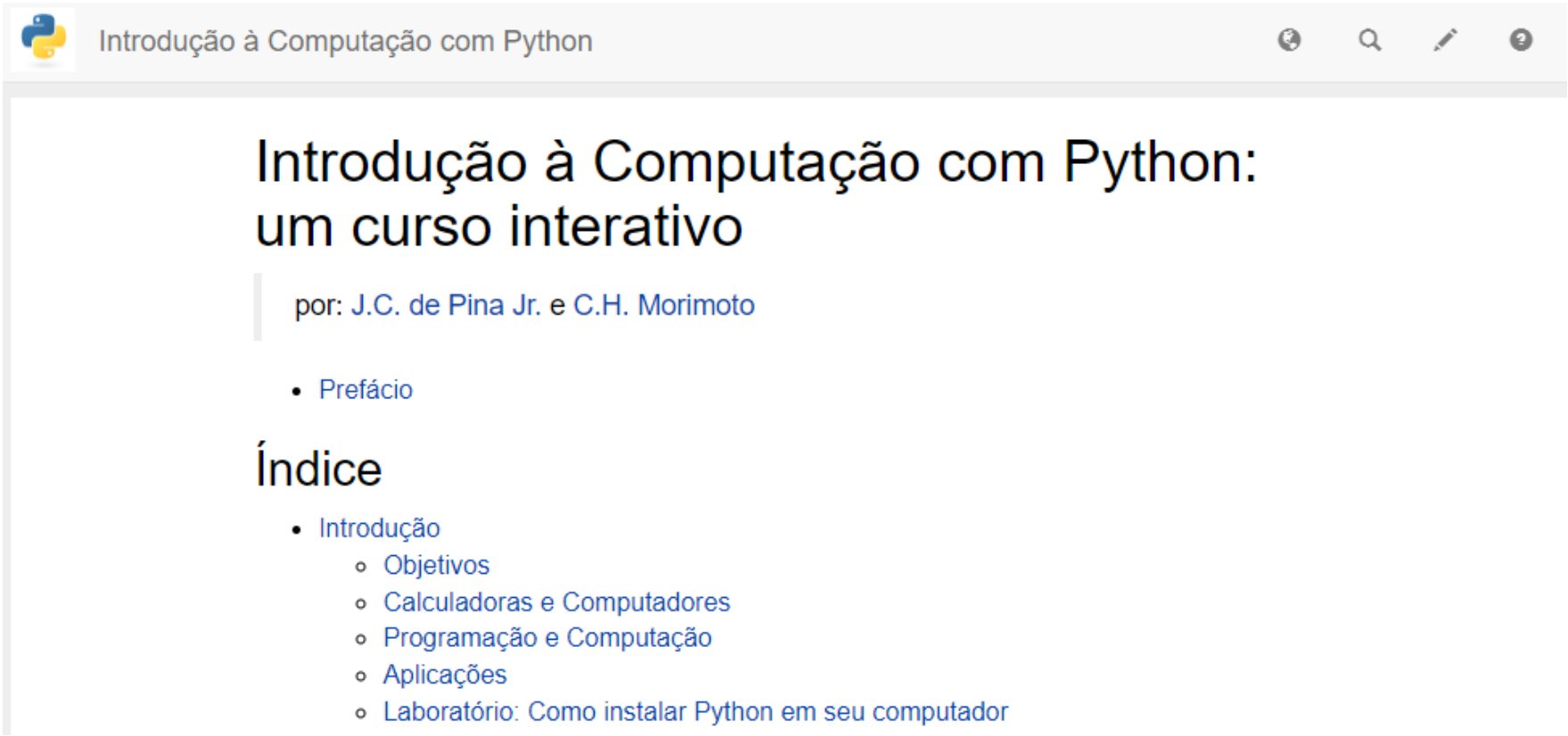
O livro em pdf é gratuito e pode ser encontrado no site da editora:

https://greenteapress.com/thinkpython/thinkCSpy/thinkCSpy.pdf



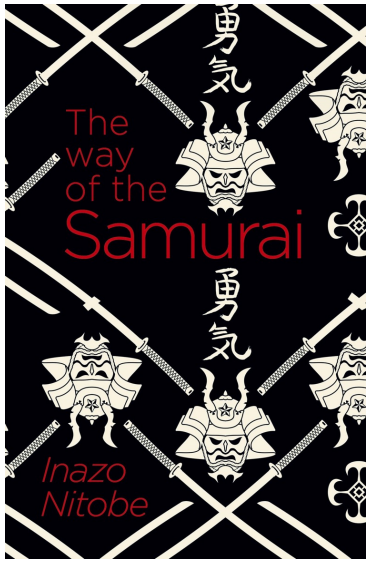
Os Profs. Coelho e Morimoto do MAC/IME disponibilizam a versão HTML em português.

Introdução à Computação com Python: um curso interativo
(https://panda.ime.usp.br/cc110/static/cc110/index.html)



Chapter 1
The way of the program

(“ O Caminho do Programa”)



1900

Objetivo da Disciplina

- Introduzir noções de programação e como utilizá-la para atividades de modelagem computacional.

Objetivos da Referência Selecionada

- Ensinar como pensar como um cientista da computação, que é uma forma de pensamento que combina alguns dos principais aspectos da matemática, engenharia e ciências naturais (e atuariais).

- A principal a habilidade de um cientista da computação é a solução de problemas – a habilidade de formular, criativamente pensar em soluções, e expressar essas soluções de forma clara e precisa.

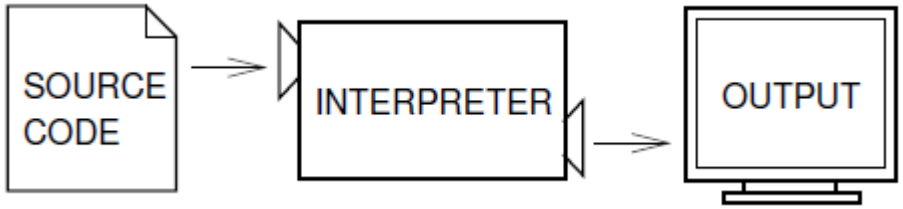
- As linguagens de programação são linguagens formais (não ambíguas) que permitem implementar essas soluções.

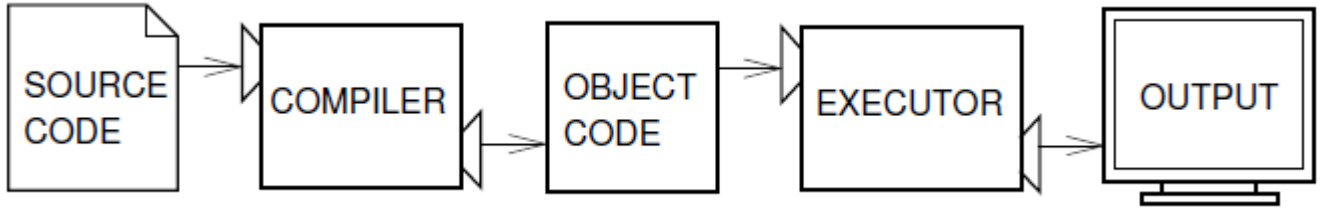
- A programação é a expressão precisa do processo de solução e, dessa forma, uma ferramenta fundamental para qualquer atividade profissional.

1.1 The Python programming language

("A Linguagem de Programação Python")

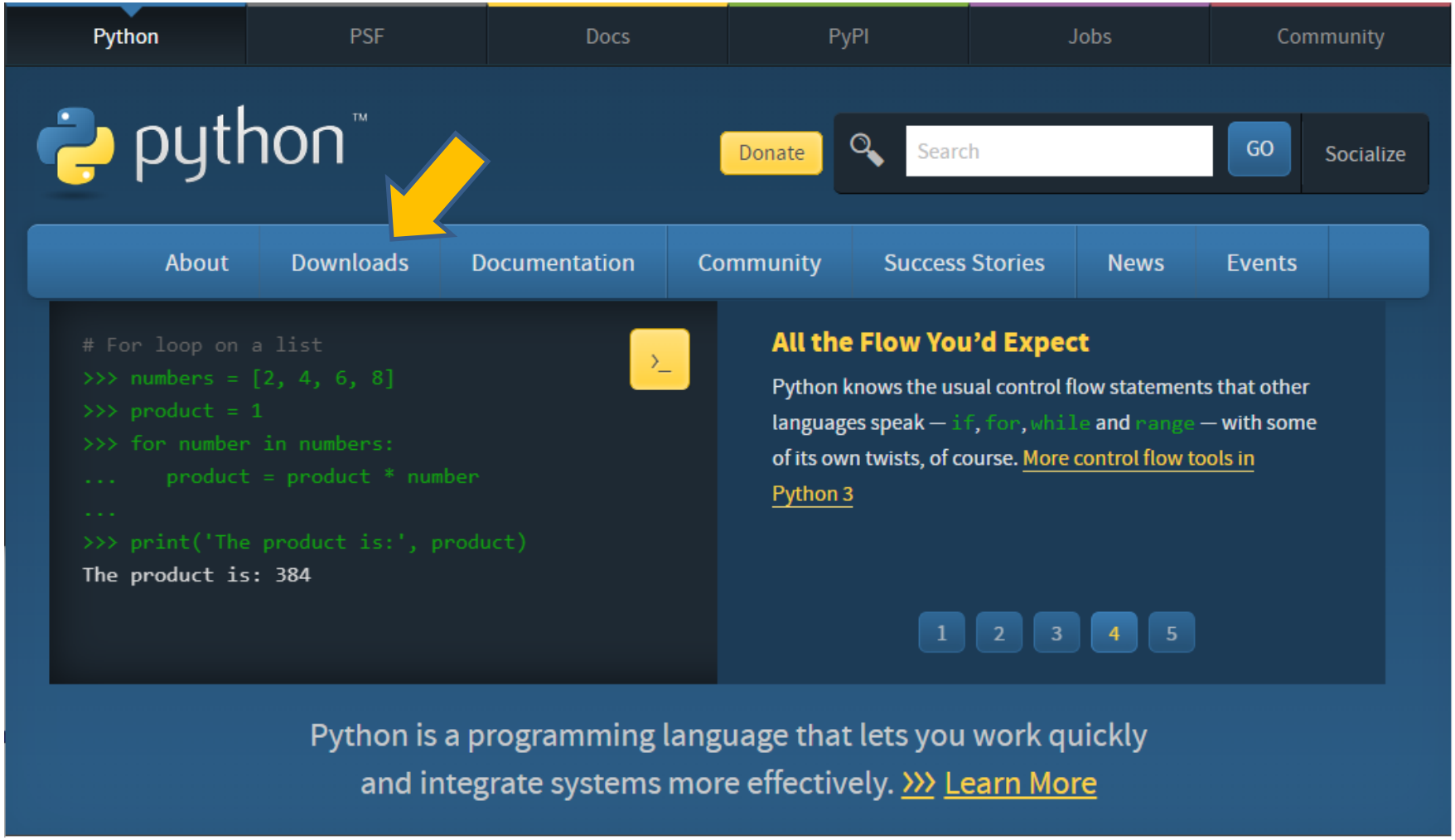
- A linguagem Python é uma linguagem de alto nível interpretada, ou seja, o código fonte é traduzido por um interpretador que gera as instruções de baixo nível que são processadas pela CPU e cujas saídas retornam através do interpretador sem a necessidade de um arquivo compilado.





Outras linguagens: C, C++, etc..

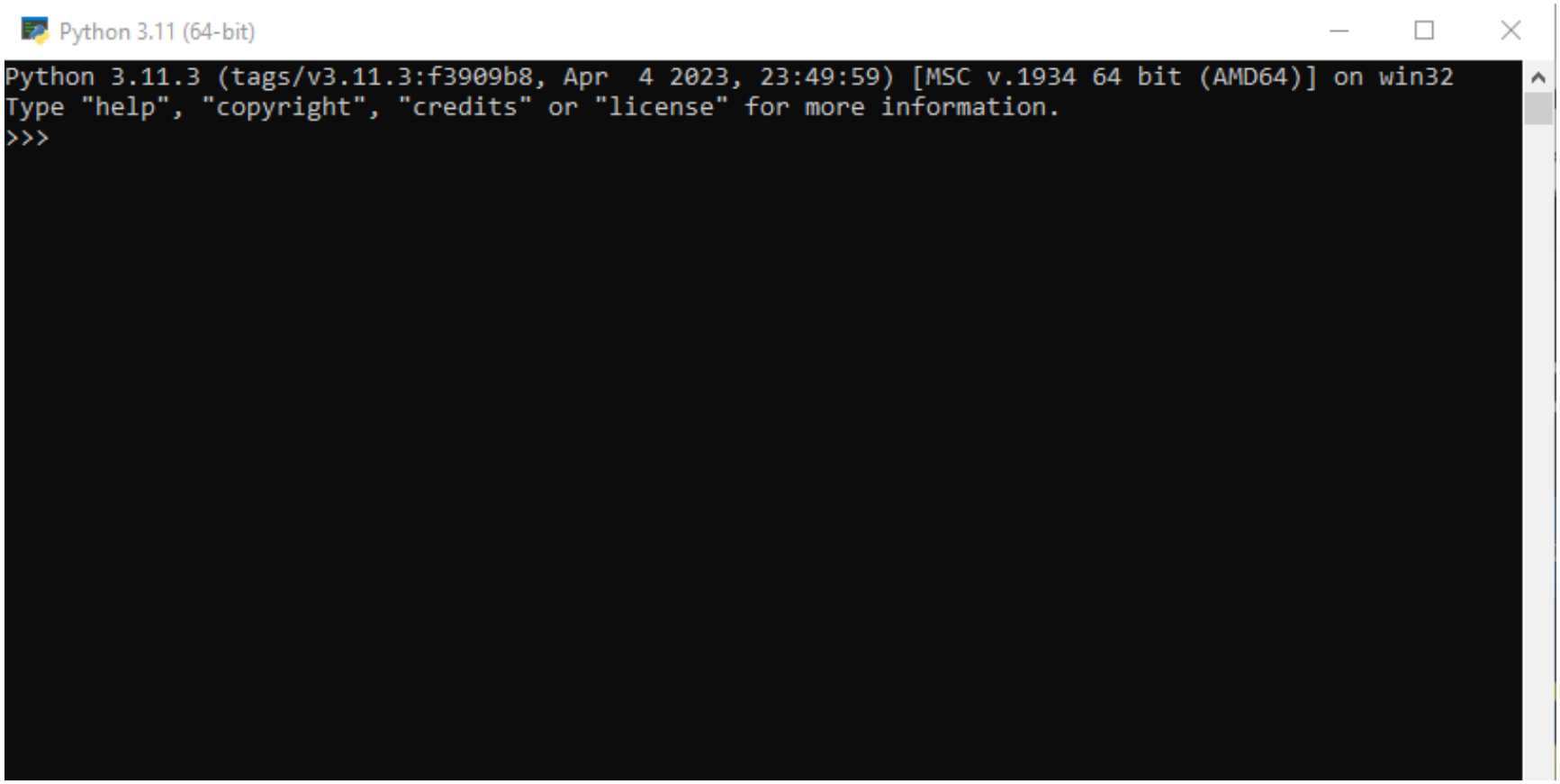
https://www.python.org/



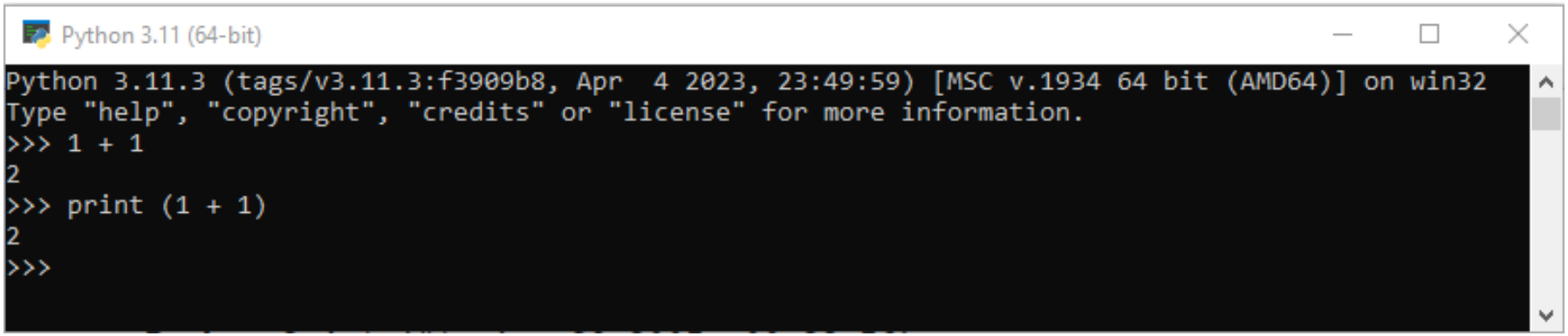
Clicando em downloads passa-se para a página com as opções de python

Automaticamente o sistema operacional da minha máquina foi identificado e a opção selecionada (pode-se obter outras distribuições selecionando abaixo)

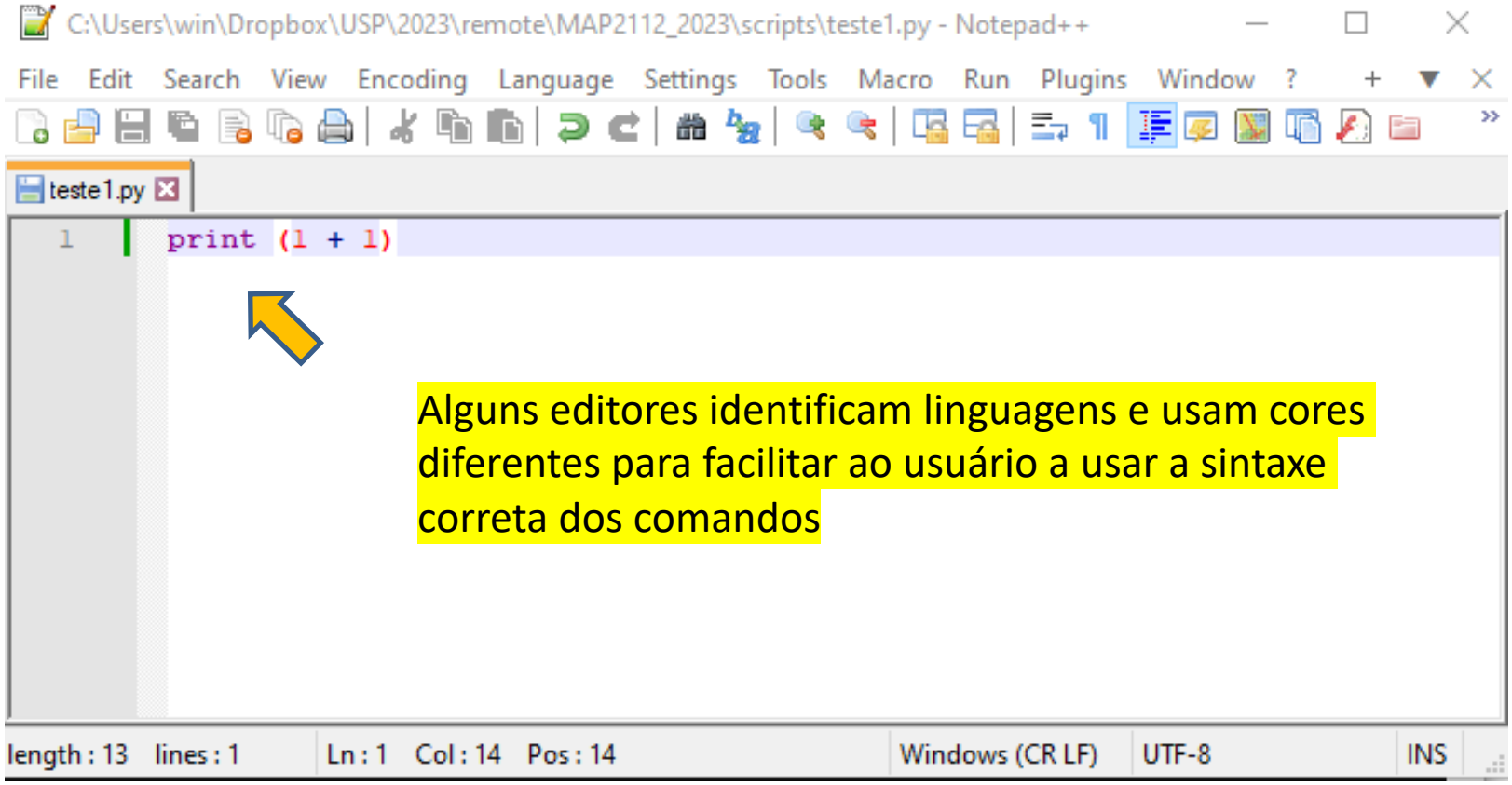
Executando o aplicativo de python instalado tem-se uma tela de execução



Como python é interpretado ele pode ser acionado comando a comando, ou ser usado para executar um script (arquivo) com a sequência de comandos.



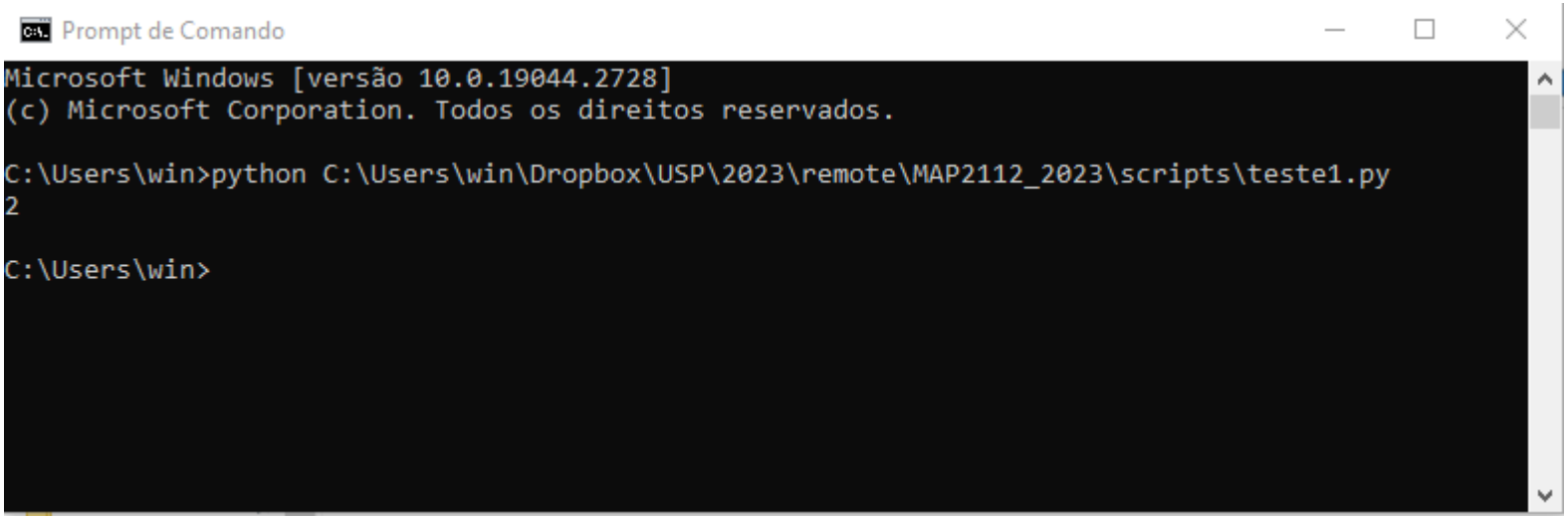
A execução direta na linha de comando (shell) do python executa as operações e comandos da linguagem



Alguns editores identificam linguagens e usam cores diferentes para facilitar ao usuário a usar a sintaxe correta dos comandos

Usando um editor de texto pode-se criar um script python

Abre-se uma shell do DOS e nela se executa o script invocando o python



O caminho completo foi utilizado para facilitar a localização mas tudo pode ser customizado para reduzir esse esforço.

1.2 What is a program?

(“ O que é um Programa ?”)

Um programa é uma sequência de instruções que especifica como realizar a computação.

A computação pode ser matemática, ou a manipulação de texto, produção de imagens ou qualquer atividade que possa ser traduzida pelo sistema de instruções.

Independentemente da linguagem alguns elementos básicos são comuns:

- Input/Entrada: Dados digitados por teclado, arquivo ou outro dispositivo
- Output/Saída: Exibir dados na tela, enviar para arquivo ou outro dispositivo
- Cálculos: Realizar operações matemáticas (de complexidade diversa)
- Execução Condicional: Verificar alguns condições para decidir qual sequência de comandos utilizar
- Repetição (Laços): Executar um conjuntos de ações repetidamente até alcançar alguma condição

1.3 What is debugging?

(“ O que é o debugar ?”)

Bug de software é um erro ou falha que ocorre num sistema ou programa de computador, resultando num comportamento incorreto, inesperado ou fora do que tenha sido pretendido pelo desenvolvedor.

Tipos de Erros:

Sintaxe – Violação das regras sintáticas da linguagem. Ex: o comando de exibição já visto é print() todo escrito em minúsculas e com o objeto a ser “impresso” entre parênteses. Escrever em maiúsculas leva a um erro de sintaxe.

```
Python 3.11 (64-bit)
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> PRINT(1+1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'PRINT' is not defined
>>> Print(1+1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined. Did you mean: 'print'?
>>>
```

Tipos de Erros (continuação):

Execução – Esses erros não aparecem antes do programa ser rodado. Eles são produzidos por exceções (Exceptions) que são violações de alguma regra do comandos da linguagem. Exemplo: Divisão por zero.

Semânticos – São erros que não são capturados na sintaxe e nem na execução. Simplemente são o uso de forma errada de comandos escritos em forma correta. É não traduzir o problema a ser resolvido corretamente embora as regras da linguagem tenham sido seguidas. Exemplo: um programa que deveria identificar números pares numa lista que tem como saída os números ímpares.

Debugging Experimental – A investigação dos erros é uma habilidade que aumenta com a experiência do programador. É checar a consistência dos comandos em testes isolados, é imprimir resultados intermediários para verificar se o programa está na direção certa do que se deseja. É ter boas práticas de codificação para evitar situações inadvertidas.

Caso tenham interesse leiam o tópico 1.4 sobre linguagens formais e naturais mas ele não é essencial.

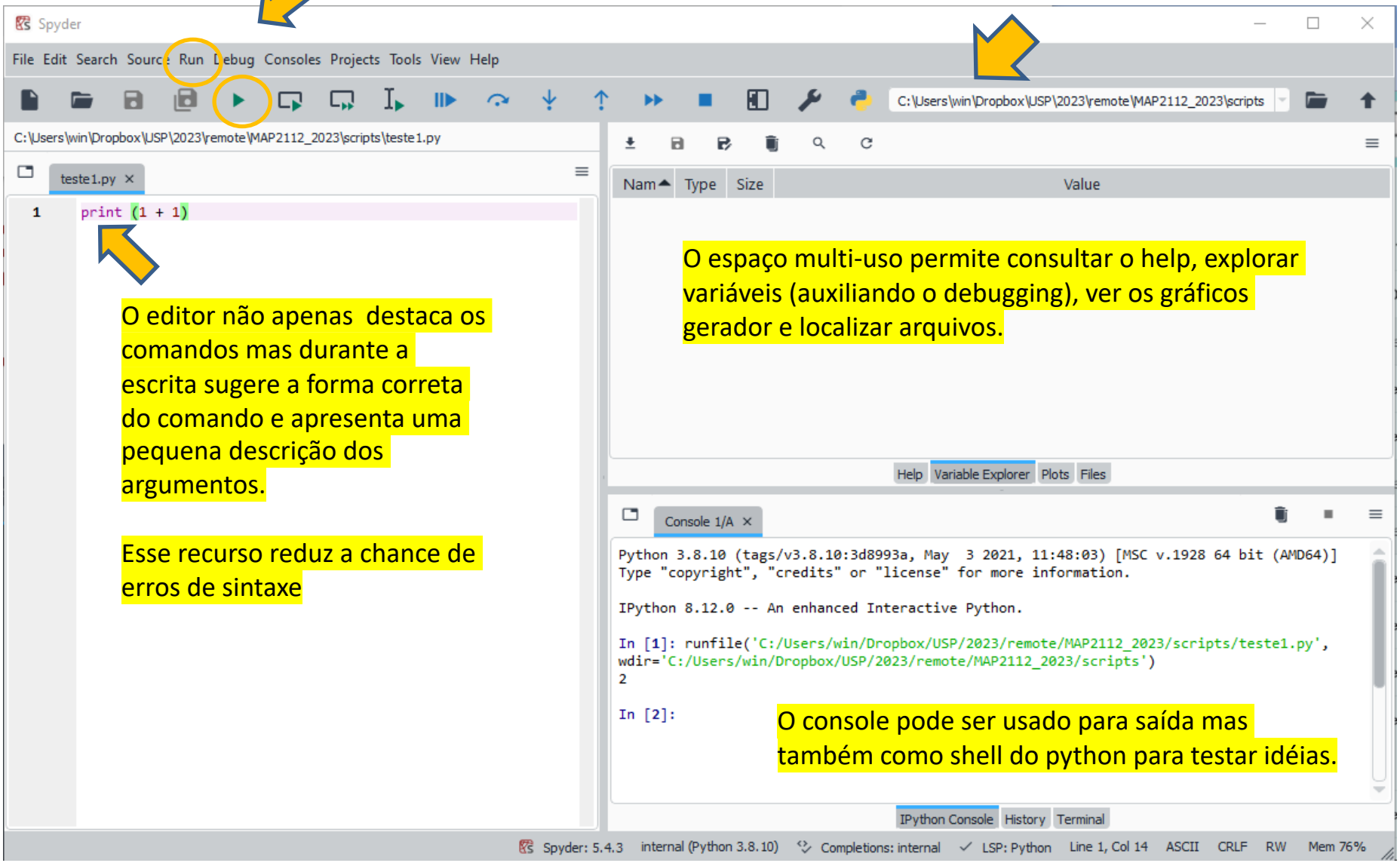
1.4 Formal and natural languages

IDE, do inglês ***Integrated Development Environment*** ou **Ambiente de Desenvolvimento Integrado**, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo

https://www.spyder-ide.org/

A execução pode ser feita pelo menu ou pela seta

O local em que se deseja que os arquivos sejam lidos e escritos pode ser definido na interface



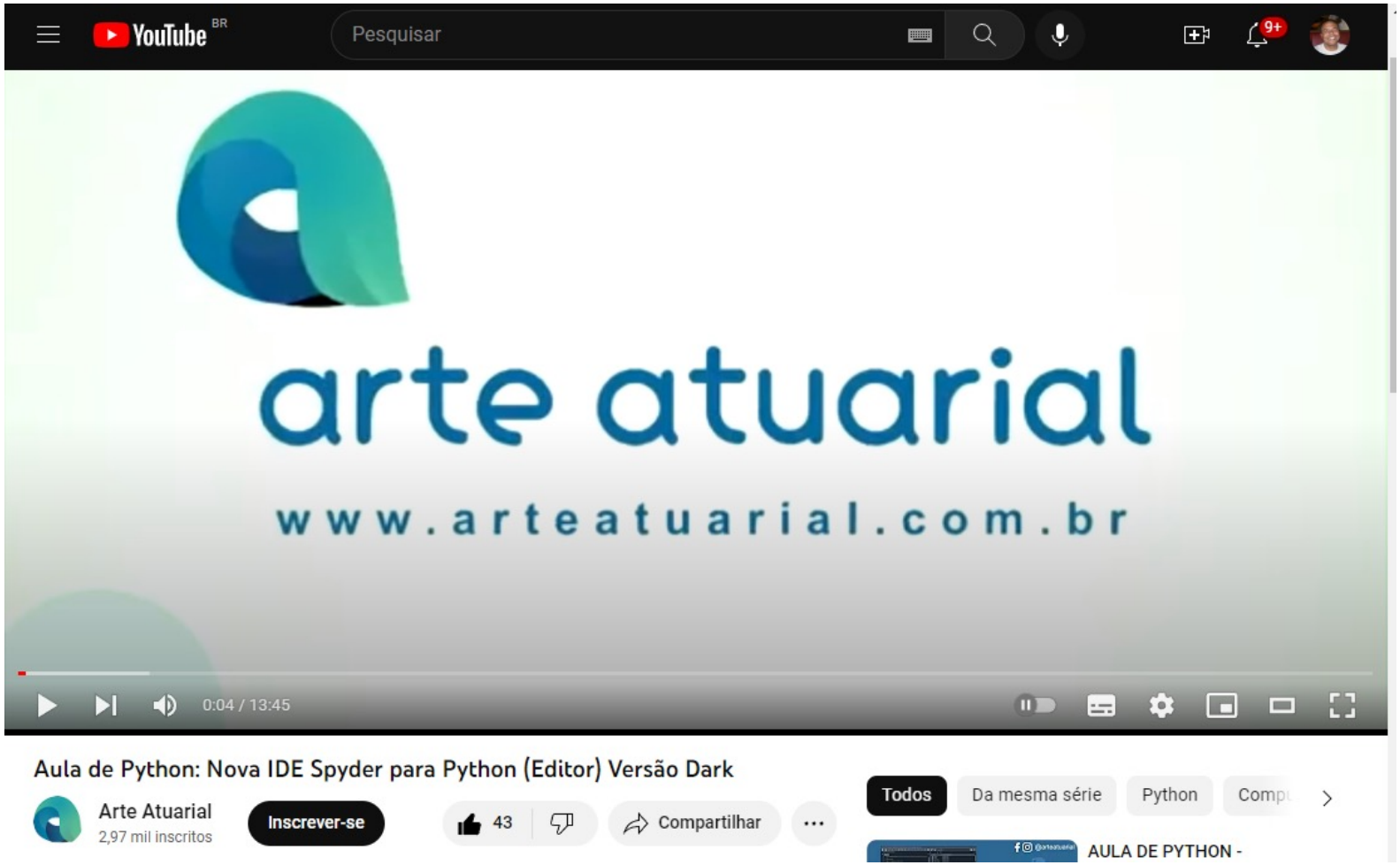
O editor não apenas destaca os comandos mas durante a escrita sugere a forma correta do comando e apresenta uma pequena descrição dos argumentos.

Esse recurso reduz a chance de erros de sintaxe

O espaço multi-uso permite consultar o help, explorar variáveis (auxiliando o debugging), ver os gráficos gerador e localizar arquivos.

O console pode ser usado para saída mas também como shell do python para testar idéias.

Para instalação e uso inicial do Spyder recomendo o vídeo:

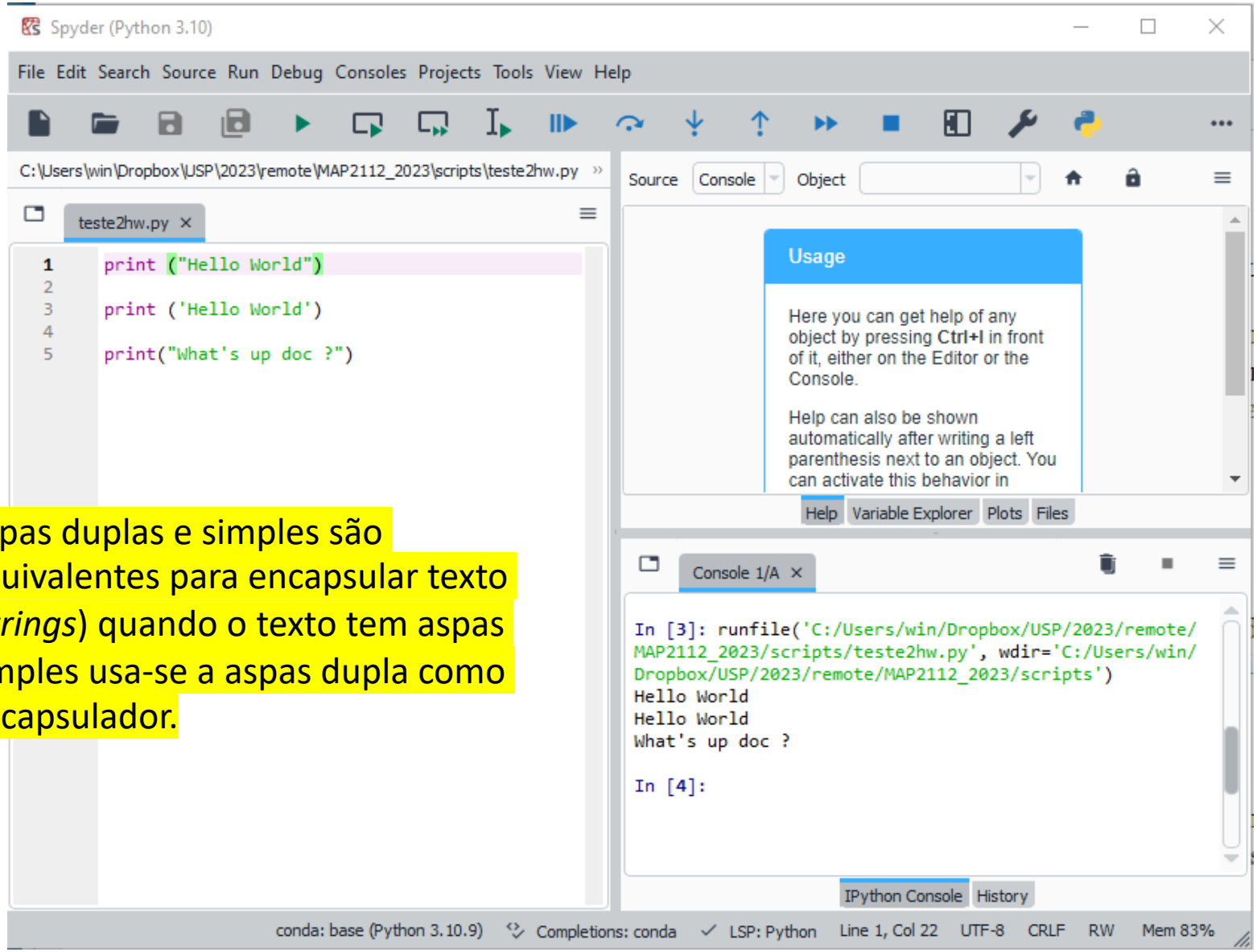


https://www.youtube.com/watch?v=GLK97IGxw6Y

1.5 The first program

("O primeiro programa")

Uma versão do clássico "Hello World!"



Aspas duplas e simples são equivalentes para encapsular texto (*strings*) quando o texto tem aspas simples usa-se a aspas dupla como encapsulador.

Chapter 2
Variables, expressions and statements

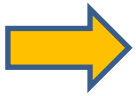
(“ Variáveis, Expressões e Comandos”)

2.1 Values and types

(“ Valores e tipos”)

Os exemplos apresentaram dois tipos de valores

```
print(1+1)

print("Hello World!")
```



```
2
Hello World!
```

Nesse caso um dos valores é um *inteiro* e o outro é uma “*string*”.

Existe uma função que interroga qual é o tipo de um valor:

```
In [7]: type("Hello World!")
Out[7]: str

In [8]: type(2)
Out[8]: int
```

string

inteiro

Números decimais pertencem ao tipo *float* (números de ponto flutuante)

```
In [9]: type(3.2)
Out[9]: float

In [10]: type(1e-06)
Out[10]: float
```

Lembre-se que o uso de aspas indica que tudo que está entre as aspas é texto.

```
In [11]: type("3.2")
Out[11]: str

In [12]: type("1e06")
Out[12]: str
```

2.2 Variables

(“ Variáveis”)

Imagine que variáveis são contêineres nomeados que podem conter valores

Nome de Variável	=	Valor a ser atribuído a variável

(operador de atribuição)

Considere os exemplos:

```
message = "What's up doc ?"

n = 17

pi = 3.14159
```

O comando print exibe o conteúdo da variável

```
In [23]: print(message)
What's up doc ?

In [24]: print(n)
17

In [25]: print(pi)
3.14159
```

Quando uma variável é criada o primeiro valor atribuído a ela define o seu tipo

```
In [17]: type(message)
Out[17]: str

In [18]: type(n)
Out[18]: int

In [19]: type(pi)
Out[19]: float
```

2.3 Variable names and keywords

(“ Nomes de Variáveis e palavras-chave”)

A escolha dos nomes das variáveis está em geral associado ao problema que está sendo resolvido, sendo uma forma auxiliar de documentação.

Os nomes podem ser arbitrariamente longos, mas uma prática comum é usar várias palavras separadas pelo caractere (_) *underscore*.

Exemplos:

```
In [26]: my_name = 'Luis Carlos'

In [27]: price_of_tea_in_China = 10

In [28]: print(my_name)
Luis Carlos

In [29]: print(price_of_tea_in_China)
10
```

Se o nome escolhido estiver fora das regras da linguagem (ilegal) o interpretador indicará um erro de sintaxe.

```
In [30]: 76trombones = 'big parade'
  Cell In[30], line 1
    76trombones = 'big parade'
     ^
SyntaxError: invalid decimal literal
```

Variáveis não podem começar com números

```
In [34]: day$ = 10
  Cell In[34], line 1
    day$ = 10
       ^
SyntaxError: invalid syntax
```

O sinal de $ no nome da variável não é um caractere permitido

```
In [32]: class = 'Computer Science 101'
  Cell In[32], line 1
    class = 'Computer Science 101'
          ^
SyntaxError: invalid syntax
```

Aparentemente não é claro porque.

A linguagem Python tem um conjunto para palavras-chave que são reservadas ao interpretador por serem comandos definidos.

Na versão 3.11 do Python são 33.

and	else	in	return
as	except	is	True
assert	finally	lambda	try
break	false	nonlocal	with
class	for	None	while
continue	from	not	yield
def	global	or	
del	if	pass	
elif	import	raise	

Se o interpretador reclama da sintaxe da variável por nenhum motivo aparente isso pode ser indício de uso de palavra reservada.

2.4 Statements

("Comandos")

Até agora foram vistos alguns comandos e funções (print, =, type). Um script python é uma sequência de comandos que modificam os valores das variáveis de acordo com essa sequência

untitled0.py*

```
1  print (1)
2  x = 2
3  print (x)
4  x = 'Hello World!'
5  print (x)
```

```
In [38]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/untitled0.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
1
2
Hello World!
```

Perceba que o comando de atribuição não produz saída/output

2.5 Evaluating expressions

("Avaliando Expressões")

Uma expressão é uma combinação valores, variáveis e operadores.

2.6 Operators and operands

("Operadores e operandos")

Operadores são símbolos que produzem ações que combinam de alguma forma valores e variáveis. Os operadores aritméticos são o melhor exemplo:

```
20+32    hour-1    hour*60+minute    minute/60    5**2    (5+9)*(15-7)
```

soma
+

subtração
-

multiplicação
*

divisão
/

exponenciação
**

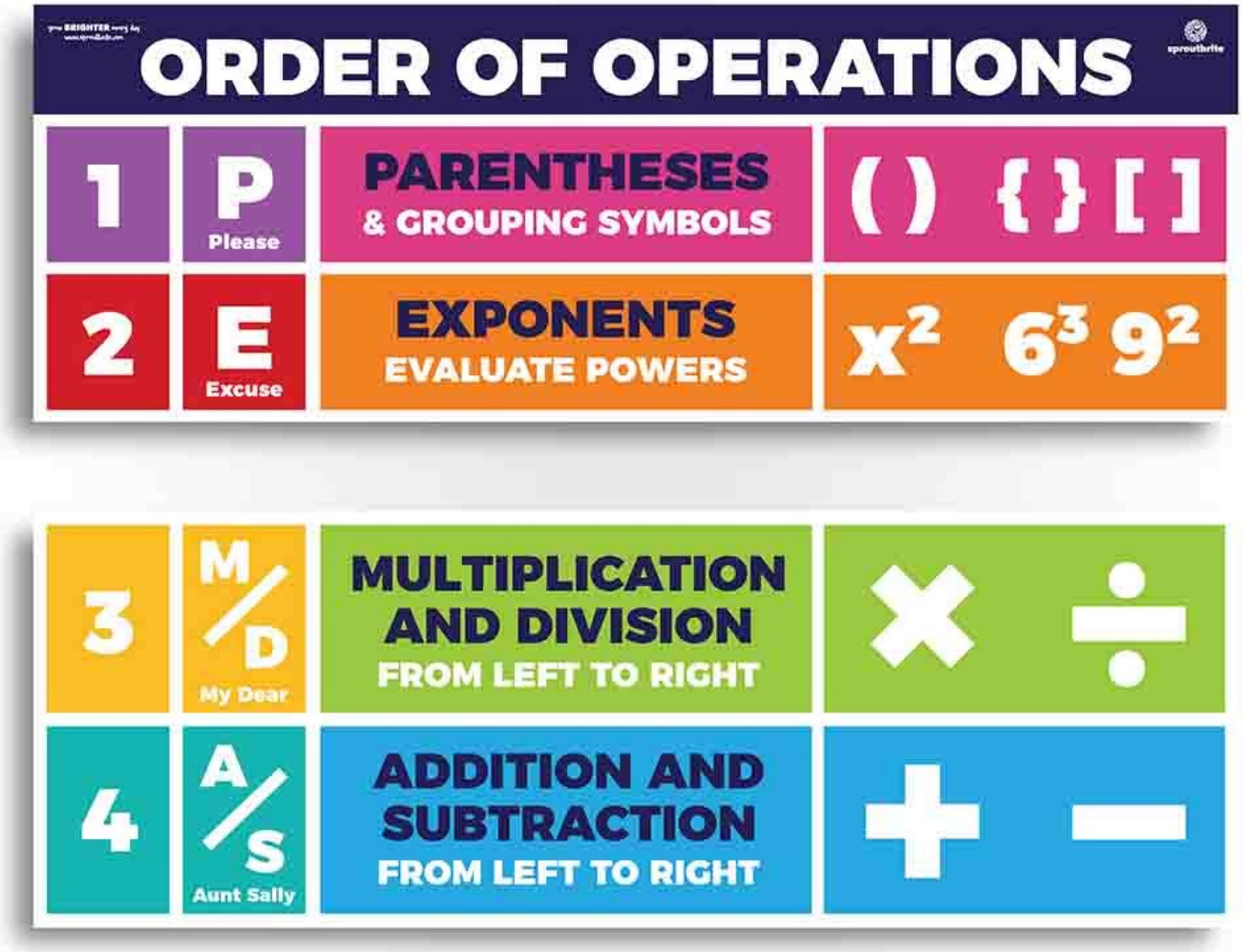
()
auxiliam na ordem de aplicação dos operadores

```
In [39]: minute = 59

In [40]: minute/60
Out[40]: 0.9833333333333333

In [41]: minute//60
Out[41]: 0

In [42]: type(minute)
Out[42]: int
```

Apesar da variável minute ter sido criada como inteira
O operador (/) produz um resultado float.
Para realizar a divisão inteira o operador é (//)

2.7 Order of operations

("Ordem das Operações")



Uma boa prática é não economizar nos parênteses para garantir que as operações estejam na ordem desejada

2.8 Operations on strings

("operações em *strings*")

Algumas operações sobre *strings* são representadas por operadores aritméticos:

```
In [43]: fruit = 'banana'

In [44]: bakedGood = ' nut bread'

In [45]: print(fruit+bakedGood)
banana nut bread
```

O operador soma (+) concatena as *strings*

```
In [46]: test = 'happy'

In [47]: 3*test
Out[47]: 'happyhappyhappy'
```

O operador multiplicação (*) repete as *strings*

2.9 Composition

("Composição")

Uma propriedade natural em programação é combinar os elementos isolados para produzir o resultado desejado:

untitled0.py*

```
1  hour = 20
2  minute = 33
3  print('No of hours since midnight: ',hour*60+minute)
4
```

```
In [48]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/untitled0.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
No of hours since midnight:  1233
```

2.10 Comments

("Comentários")

A medida que os programas crescem (ou o tempo passa desde a implementação inicial) torna-se difícil saber apenas pela leitura do código o que está sendo realizado.

Uma boa prática de programação é a inclusão de comentários, usando o hashtag (#) ou trelha em português.

As duas formas a seguir são adequadas:

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

```
percentage = (minute * 100) / 60     # caution: integer division
```

Chapter 3 Functions

3.1 Function calls

(“ Chamada de funções”)

Já vimos o uso de funções quando interrogamos os tipos de variáveis (e valores)

```
In [1]: type(32)
Out[1]: int

In [2]: type("32")
Out[2]: str
```

O resultado da aplicação de funções pode ser atribuído a uma variável. Ex:

```
In [3]: betty = type("32")

In [4]: print (betty)
<class 'str'>
```

3.2 Type conversion

("Conversão de Tipo")

Existe um conjunto de funções que permitem converter variáveis ou valores de tipo. Alguns exemplos.

Como o conteúdo do string é um inteiro não há problema na conversão

```
In [5]: int("32")
Out[5]: 32

In [6]: int("Hello")
Traceback (most recent call last):

  Cell In[6], line 1
    int("Hello")

ValueError: invalid literal for int() with base 10: 'Hello'
```

No caso de um string genérico o python não sabe o que fazer indicando um erro

No caso de inteiros e ponto flutuante a conversão é mais natural.

```
In [7]: int(3.9999)
Out[7]: 3

In [8]: int(-2.3)
Out[8]: -2
```

Os valores de ponto flutuante são truncados e não arredondados

Existe uma função de arredondamento

```
In [9]: round(3.9999)
Out[9]: 4
```

round

Definition : round(...)

Type : Function of builtins module

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None. Otherwise the return value has the same type as the number. ndigits may be negative.

A função *float* converte inteiros e strings para números de ponto flutuante.

```
In [10]: float(32)
Out[10]: 32.0

In [11]: float("3.24159")
Out[11]: 3.24159
```

A função *str* converte inteiros e números de ponto flutuante para strings.

```
In [12]: str(32)
Out[12]: '32'

In [13]: str(3.14159)
Out[13]: '3.14159'
```

3.3 Type coercion

O exemplo de coerção de tipo da referência não se aplica mais ao pyhton 3.0

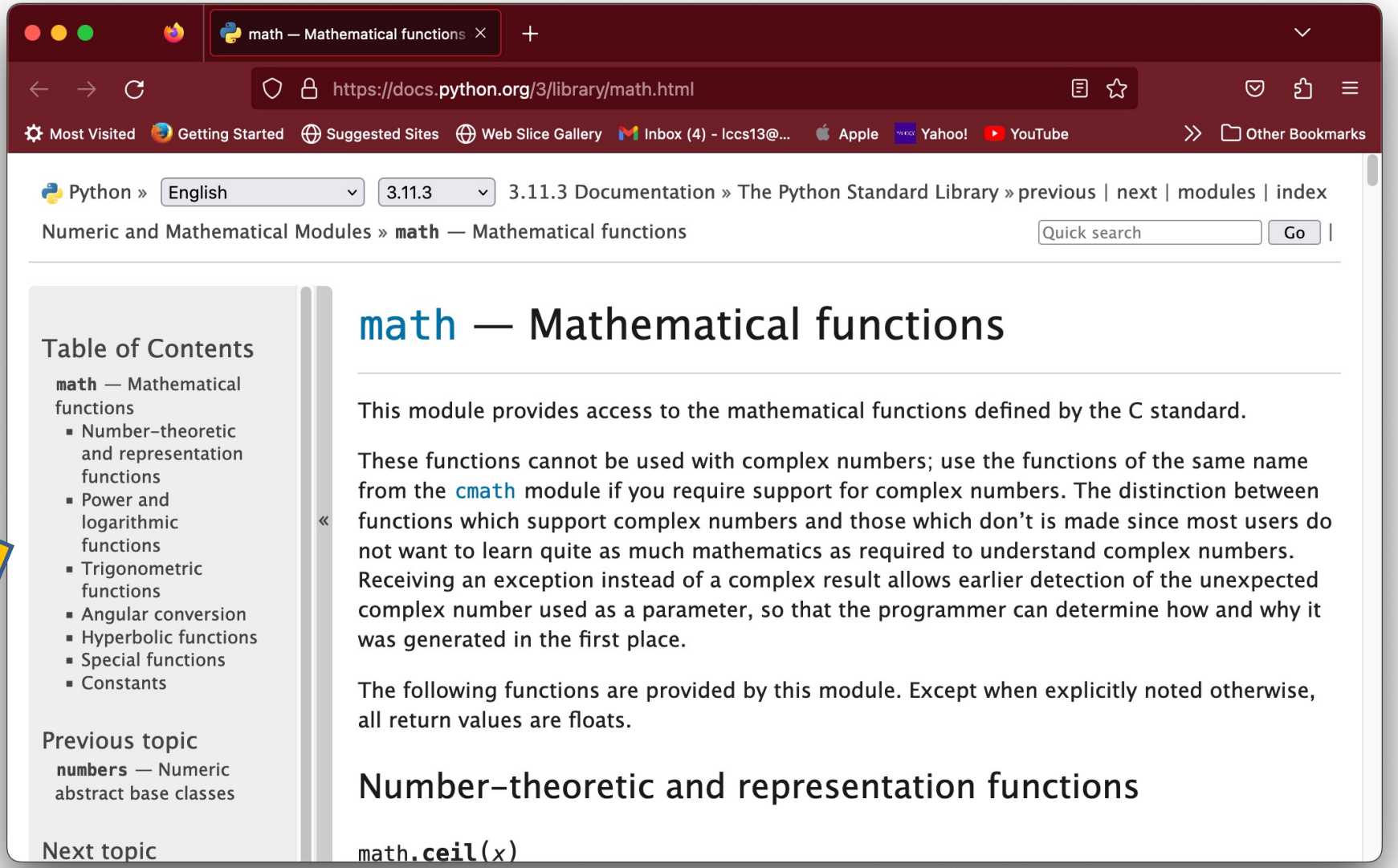
3.4 Math functions

("Funções Matemáticas")

As funções matemáticas do python não fazem parte do módulo básico. Para invocá-las é necessário carregar o módulo math.

Um módulo é um arquivo que contém uma coleções de funções relacionadas

```
In [16]: sin(pi/2)
Traceback (most recent call last):

  Cell In[16], line 1
    sin(pi/2)

NameError: name 'sin' is not defined


In [17]: import math

In [18]: sin(pi/2)
Traceback (most recent call last):

  Cell In[18], line 1
    sin(pi/2)

NameError: name 'sin' is not defined


In [19]: math.sin(pi/2)
Traceback (most recent call last):

  Cell In[19], line 1
    math.sin(pi/2)

NameError: name 'pi' is not defined


In [20]: math.sin(math.pi/2)
Out[20]: 1.0
```


math — Mathematical functions
https://docs.python.org/3/library/math.html
Most Visited
Getting Started
Suggested Sites
Web Slice Gallery
Inbox (4) - lccs13@...
Apple
Yahoo!
YouTube
Other Bookmarks
Python »
English
3.11.3
3.11.3 Documentation » The Python Standard Library » previous | next | modules | index
Numeric and Mathematical Modules » math — Mathematical functions
Quick search
Go
Table of Contents
math — Mathematical functions
Number-theoretic and representation functions
Power and logarithmic functions
Trigonometric functions
Angular conversion
Hyperbolic functions
Special functions
Constants
Previous topic
numbers — Numeric abstract base classes
Next topic
math — Mathematical functions
This module provides access to the mathematical functions defined by the C standard.
These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.
The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.
Number-theoretic and representation functions
math.ceil(x)

Trigonometric functions ¶

`math.`**`acos`**`(x)`

Return the arc cosine of *x*, in radians. The result is between `0` and `pi`.

`math.`**`asin`**`(x)`

Return the arc sine of *x*, in radians. The result is between `-pi/2` and `pi/2`.

`math.`**`atan`**`(x)`

Return the arc tangent of *x*, in radians. The result is between `-pi/2` and `pi/2`.

`math.`**`atan2`**`(y, x)`

Return `atan(y / x)`, in radians. The result is between `-pi` and `pi`. The vector in the plane from the origin to point `(x, y)` makes this angle with the positive X axis. The point of `atan2()` is that the signs of both inputs are known to it, so it can compute the correct quadrant for the angle. For example, `atan(1)` and `atan2(1, 1)` are both `pi/4`, but `atan2(-1, -1)` is `-3*pi/4`.

`math.`**`cos`**`(x)`

Return the cosine of *x* radians.

`math.`**`dist`**`(p, q)`

Return the Euclidean distance between two points *p* and *q*, each given as a sequence (or iterable) of coordinates. The two points must have the same dimension.

Roughly equivalent to:

```
sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))
```

Exemplos das funções trigonométricas

3.5 Composition

("Composição")

Funções podem ser aplicadas sobre outras funções para produzir o resultado desejado.

temp.py — Temporary file

```
1  import math
2
3  degrees = 45
4  angle = degrees * 2* math.pi/ 360.0
5  print(math.sin(angle))
6  print(math.sqrt(2)/2.0)
7
8  x = math.cos(angle + math.pi/2)
9  print(x)
10
11 x = math.exp(math.log(10.0))
12 print(x)
13
```

```
In [21]: runfile('/Users/lccs13/.spyder-py3/temp.py', wdir='/Users/lccs13/.spyder-py3')
0.7071067811865475
0.7071067811865476
-0.7071067811865475
10.000000000000002
```

Até a próxima aula procure ajustar o seu ambiente e execute os exemplos usados nessa aula.

Fim Aula 02