

Itaú Unibanco

Itaú

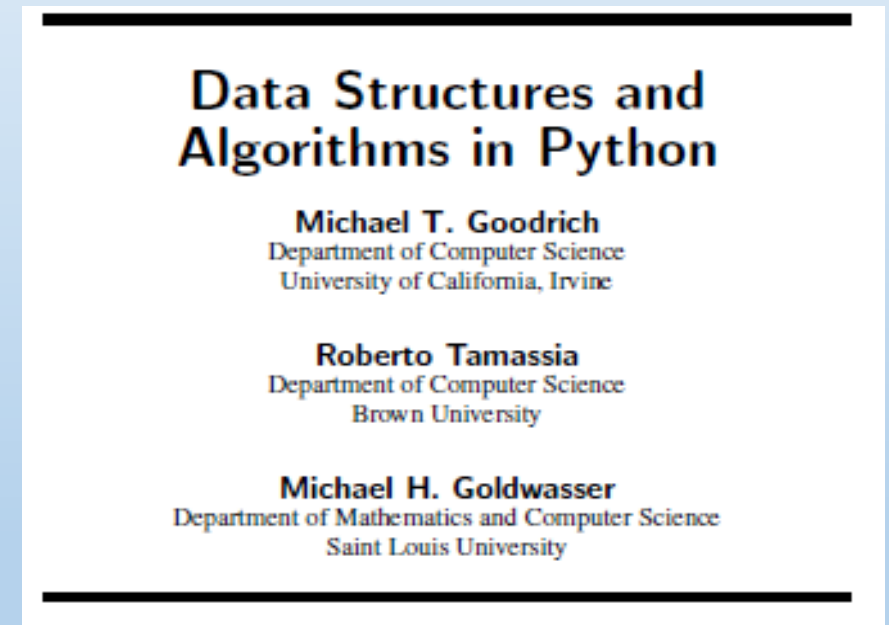
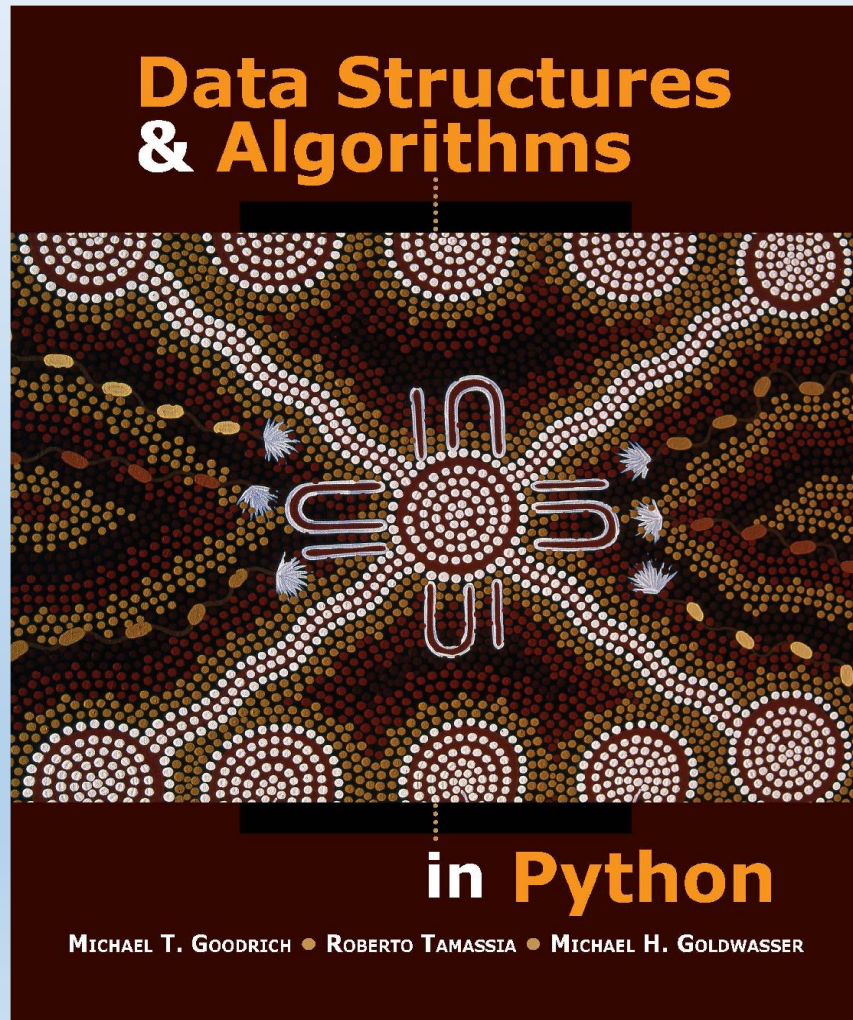
Programa de formação

ITAÚ analytics.

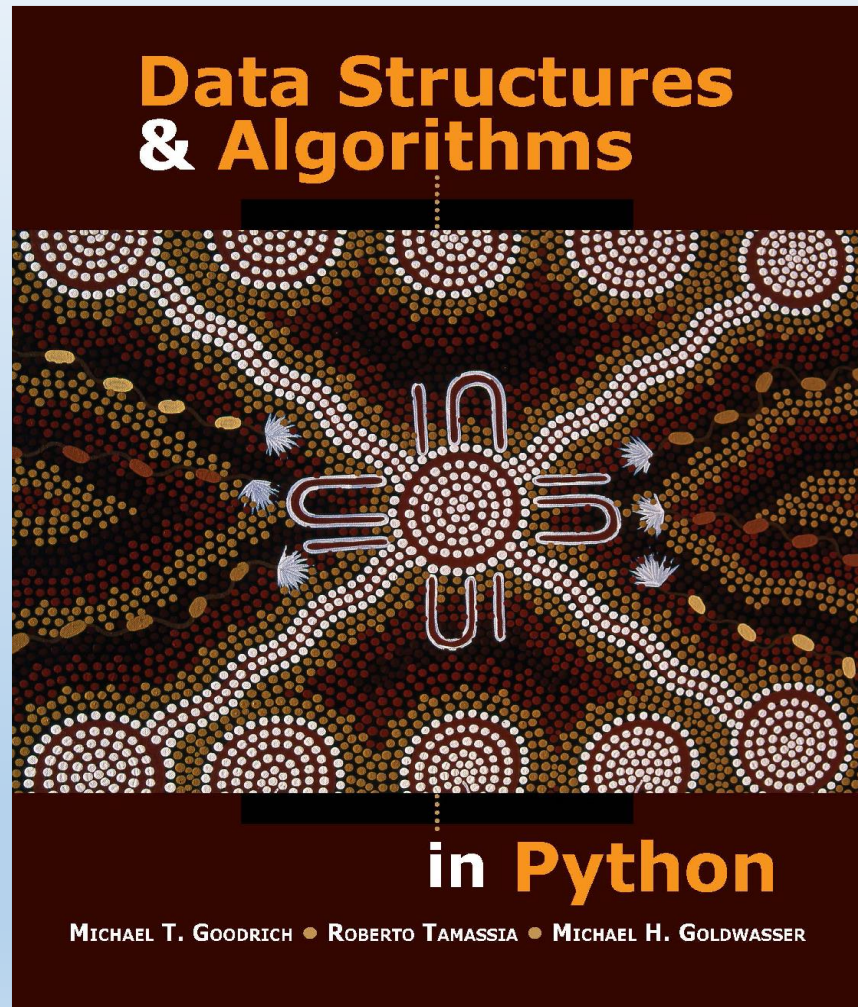


Módulo I – Fundamentos Computacionais
Ses 2 - Aula 1 – Fundamentos de Python
Prof. Dr. Luiz Alberto Vieira Dias
Prof. Dr. Lineu Mialaret

Livro Texto

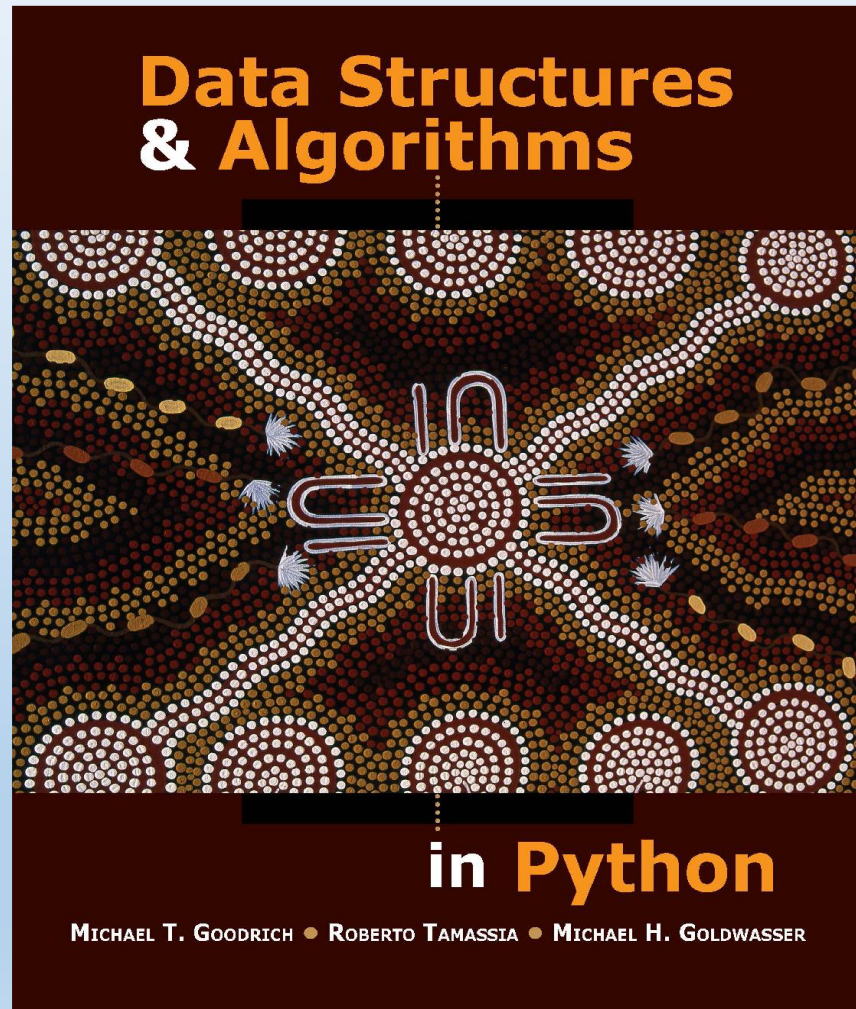


Livro Texto



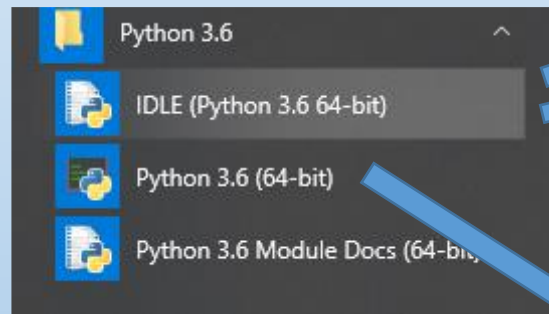
1	Python Primer	1
1.1	Python Overview	2
1.1.1	The Python Interpreter	2
1.1.2	Preview of a Python Program	3
1.2	Objects in Python	4
1.2.1	Identifiers, Objects, and the Assignment Statement	4
1.2.2	Creating and Using Objects	6
1.2.3	Python's Built-In Classes	7
1.3	Expressions, Operators, and Precedence	12
1.3.1	Compound Expressions and Operator Precedence	17
1.4	Control Flow	18
1.4.1	Conditionals	18
1.4.2	Loops	20
1.5	Functions	23
1.5.1	Information Passing	24
1.5.2	Python's Built-In Functions	28
1.6	Simple Input and Output	30
1.6.1	Console Input and Output	30
1.6.2	Files	31
1.7	Exception Handling	33
1.7.1	Raising an Exception	34
1.7.2	Catching an Exception	36
1.8	Iterators and Generators	39
1.9	Additional Python Conveniences	42
1.9.1	Conditional Expressions	42
1.9.2	Comprehension Syntax	43
1.9.3	Packing and Unpacking of Sequences	44
1.10	Scopes and Namespaces	46
1.11	Modules and the Import Statement	48
1.11.1	Existing Modules	49
1.12	Exercises	51

Livro Texto

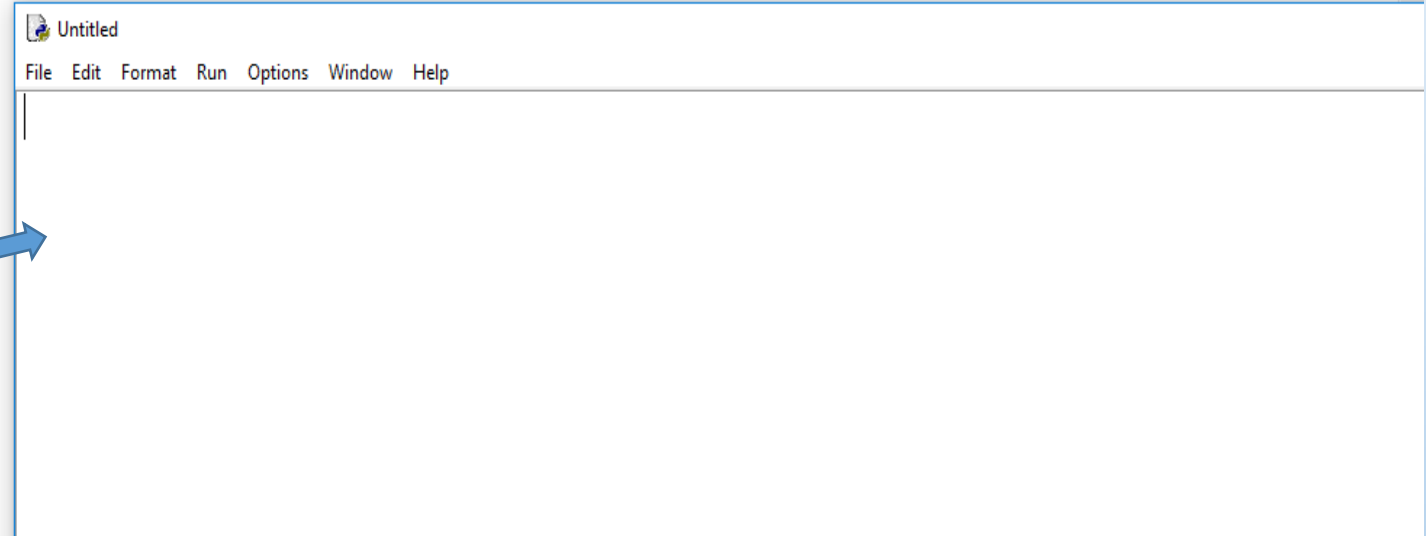


2	Object-Oriented Programming	56
2.1	Goals, Principles, and Patterns	57
2.1.1	Object-Oriented Design Goals	57
2.1.2	Object-Oriented Design Principles	58
2.1.3	Design Patterns	61
2.2	Software Development	62
2.2.1	Design	62
2.2.2	Pseudo-Code	64
2.2.3	Coding Style and Documentation	64
2.2.4	Testing and Debugging	67
2.3	Class Definitions	69
2.3.1	Example: CreditCard Class	69
2.3.2	Operator Overloading and Python's Special Methods	74
2.3.3	Example: Multidimensional Vector Class	77
2.3.4	Iterators	79
2.3.5	Example: Range Class	80
2.4	Inheritance	82
2.4.1	Extending the CreditCard Class	83
2.4.2	Hierarchy of Numeric Progressions	87
2.4.3	Abstract Base Classes	93
2.5	Namespaces and Object-Oriented	96
2.5.1	Instance and Class Namespaces	96
2.5.2	Name Resolution and Dynamic Dispatch	100
2.6	Shallow and Deep Copying	101
2.7	Exercises	103

IDLE



```
Python 3.6.5rc1 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5rc1 (v3.6.5rc1:f03c5148cf, Mar 14 2018, 03:12:11) [MSC v.1913 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```



```
Python 3.6 (64-bit)
Python 3.6.5rc1 (v3.6.5rc1:f03c5148cf, Mar 14 2018, 03:12:11) [MSC v.1913 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Hello, world")
Hello, world
>>> _
```

Visão Prévia de um Programa Python

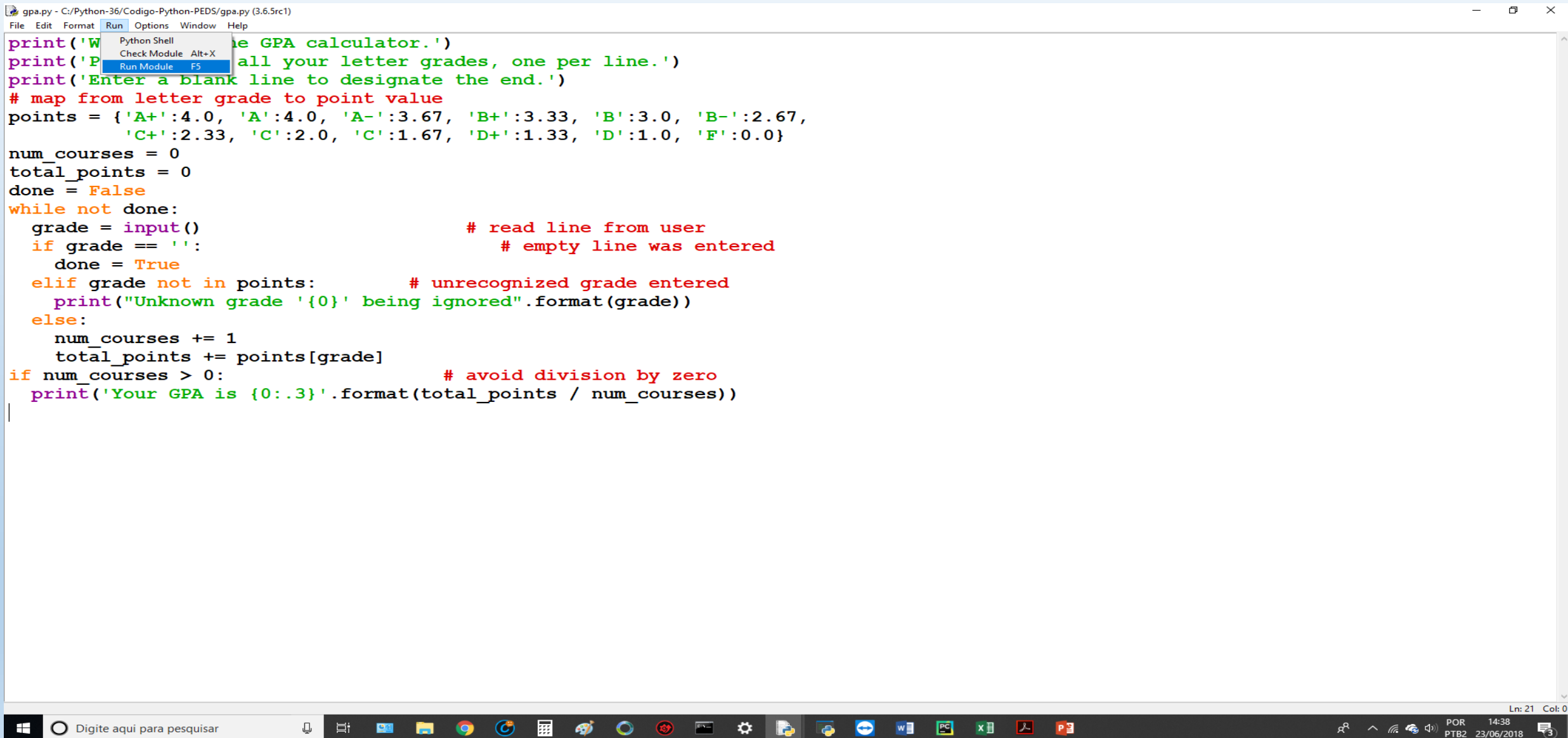
```
print('Welcome to the GPA calculator.')
print('Please enter all your letter grades, one per line.')
print('Enter a blank line to designate the end.')
# map from letter grade to point value
points = {'A+':4.0, 'A':4.0, 'A-':3.67, 'B+':3.33, 'B':3.0, 'B-':2.67,
          'C+':2.33, 'C':2.0, 'C-':1.67, 'D+':1.33, 'D':1.0, 'F':0.0}
num_courses = 0
total_points = 0
done = False
while not done:
    grade = input()          # read line from user
    if grade == '':          # empty line was entered
        done = True
    elif grade not in points: # unrecognized grade entered
        print("Unknown grade '{0}' being ignored".format(grade))
    else:
        num_courses += 1
        total_points += points[grade]
if num_courses > 0:         # avoid division by zero
    print('Your GPA is {0:.3}'.format(total_points / num_courses))
```


Visão Prévia de um Programa Python (cont)

```
gpa.py - D:/Arq-Python/gpa.py (3.6.5)
File Edit Format Run Options Window Help
print('Welcome to the GPA calculator.')
print('Please enter all your letter grades, one per line.')
print('Enter a blank line to designate the end.')
# map from letter grade to point value
points = {'A+':4.0, 'A':4.0, 'A-':3.67, 'B+':3.33, 'B':3.0, 'B-':2.67,
          'C+':2.33, 'C':2.0, 'C-':1.67, 'D+':1.33, 'D':1.0, 'F':0.0}
num_courses = 0
total_points = 0
done = False
while not done:
    grade = input()                # read line from user
    if grade == '':               # empty line was entered
        done = True
    elif grade not in points:     # unrecognized grade entered
        print("Unknown grade '{0}' being ignored".format(grade))
    else:
        num_courses += 1
        total_points += points[grade]
if num_courses > 0:              # avoid division by zero
    print('Your GPA is {0:.3}'.format(total_points / num_courses))
|
```

Ln: 21 Col: 0

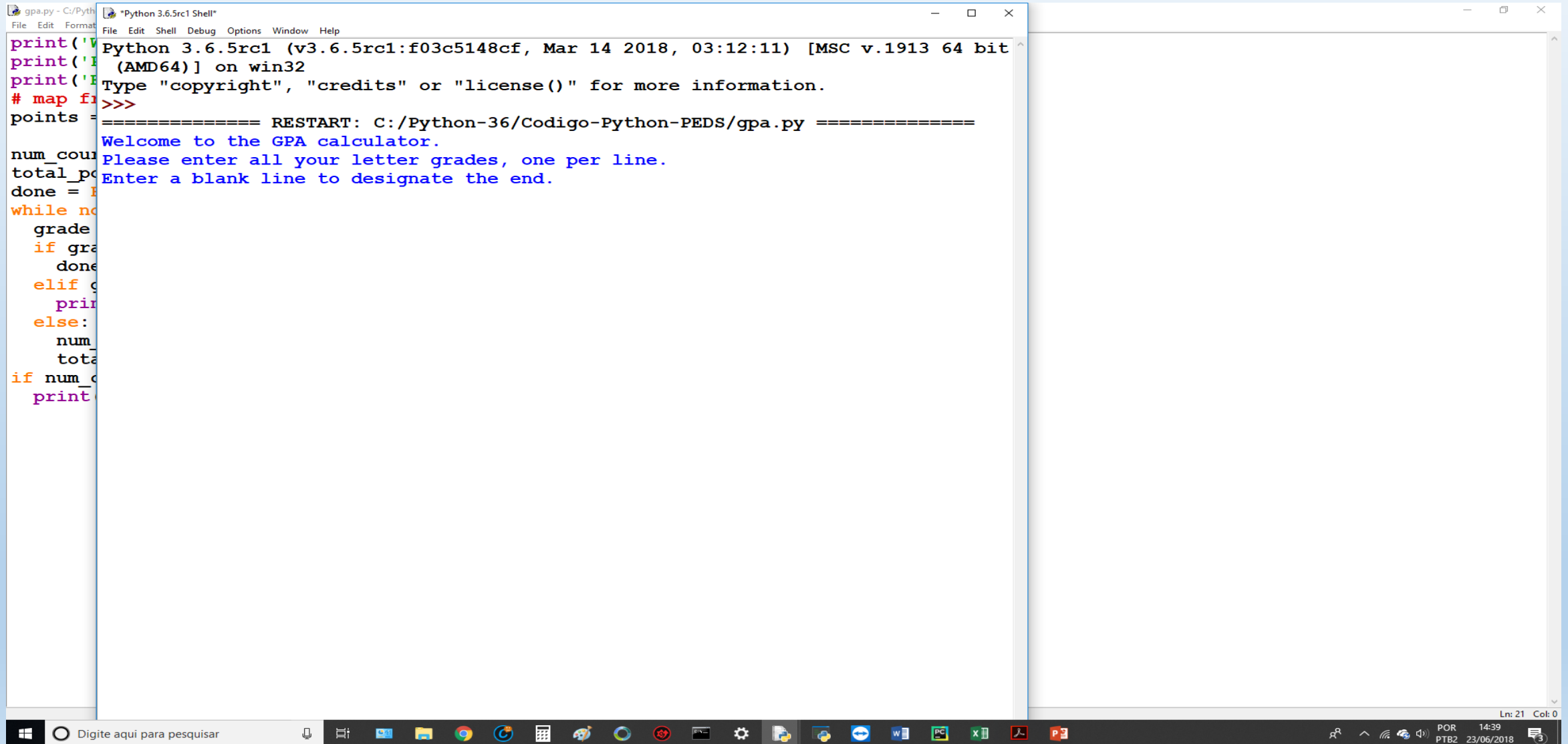
Visão Prévia de um Programa Python (cont)



```
gpa.py - C:/Python-36/Codigo-Python-PEDS/gpa.py (3.6.5rc1)
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5

print('Welcome to the GPA calculator.')
print('Please enter all your letter grades, one per line.')
print('Enter a blank line to designate the end.')
# map from letter grade to point value
points = {'A+':4.0, 'A':4.0, 'A-':3.67, 'B+':3.33, 'B':3.0, 'B-':2.67,
          'C+':2.33, 'C':2.0, 'C-':1.67, 'D+':1.33, 'D':1.0, 'F':0.0}
num_courses = 0
total_points = 0
done = False
while not done:
    grade = input()                # read line from user
    if grade == '':                # empty line was entered
        done = True
    elif grade not in points:      # unrecognized grade entered
        print("Unknown grade '{0}' being ignored".format(grade))
    else:
        num_courses += 1
        total_points += points[grade]
if num_courses > 0:                # avoid division by zero
    print('Your GPA is {0:.3}'.format(total_points / num_courses))
```


Visão Prévia de um Programa Python (cont)



The screenshot shows a Windows desktop with a Python 3.6.5rc1 Shell window open. The shell window displays the output of a Python script named gpa.py. The script is a GPA calculator that prompts the user to enter letter grades one per line. The output shows the program's version, architecture, and a restart message. The user has entered 'A' and 'B' as grades, and the program has calculated a GPA of 3.5. The shell window also shows the file explorer and taskbar at the bottom.

```
Python 3.6.5rc1 (v3.6.5rc1:f03c5148cf, Mar 14 2018, 03:12:11) [MSC v.1913 64 bit  
(AMD64)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Python-36/Codigo-Python-PEDS/gpa.py =====  
Welcome to the GPA calculator.  
Please enter all your letter grades, one per line.  
Enter a blank line to designate the end.  
A  
B  
<Enter>  
3.5  
>>>
```

Instruções no Python

- Uma **declaração** (*statement*) no Python é uma instrução que o interpretador lê e executa
 - Pode ser uma expressão ou atribuição (*assignment statement*)
- Com **expressões** podem ser realizadas operações como adição, subtração, concatenação e também pode-se ter uma expressão como uma chamada para uma função que avalia os resultados

Using arithmetic expressions

```
>>> ((10 + 2) * 100 / 5 - 200)  
40.0
```

Using function

```
>>> pow(2, 10)  
1024
```

Using function and expression

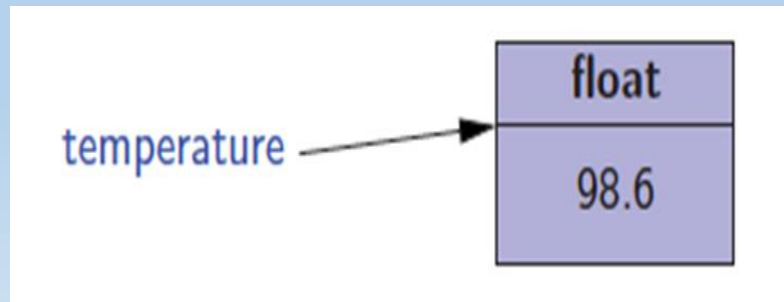
```
>>> pow((10 + 2*5) * 2, 4)  
2560000
```

Atribuição no Python

- A linguagem Python constitui-se numa linguagem orientada a objetos e as classes formam a base para todos os tipos de dados
- A instrução de atribuição (*assignment statement*) é uma das mais importantes na linguagem

```
>>> temperature = 96.6 # assignment instruction
```

- A instrução acima estabelece temperature como um identificador (*identifier*) e então o associa com o objeto representado pelo lado direito da instrução, um objeto ponto flutuante com valor 98.6



Atribuição no Python

- Pode-se ter diversos tipos de atribuições
 - Atribuição por valor literal
 - Atribuição de variáveis
 - Atribuição de expressões
 - Atribuição simultânea
 - Atribuição múltipla

```
>>> a=b=c=1      # Multiple Assignment
>>> print (a,b,c)
1 1 1
```

```
>>> a,b,c = 5,56,'João'    # Simultaneous Assignment
>>> print (a,b,c)
5 56 João
```

```
>>> test = "Learn Python" # literal assignment
>>> type (test)
<class 'str'>
>>> id(test) # return memory address
2493824010160
```

```
>>> another_test = test # variable assignment
>>> id(another_test) # return memory address
2493824010160
```

```
>>> test = 2 * 5 / 10 # expression assignment
>>> print(test)
1.0
>>> type(test) # return object type
<class 'float'>
>>> id (test)
2493790294352
```


Instruções Multilinha no Python

- Toda instrução no Python termina com o caractere de nova linha
- Pode-se estender a instrução para mais de uma linha
 - Usando a \
 - Usando (, [ou {
- Pode-se também colocar múltiplas instruções em uma única linha usando “;”

```
>>> # Using a \ for continue the multi-line statement
>>> a = 1 + 2 + 3 + \
... 4 + 5 + 6 + \
... 7 + 8 + 9
>>> a
45
```

```
# Using a ( for continue the multi-line statement
>>> a = (1 + 2 + 3 +
... 4 + 5 + 6 +
... 7 + 8 + 9)
>>> a
45
```

```
# multiple statements in a one line
>>> a = 1; b = 2; c = 3
>>> print (a,b,c,)
1 2 3
```

Instruções Multilinha no Python (cont)

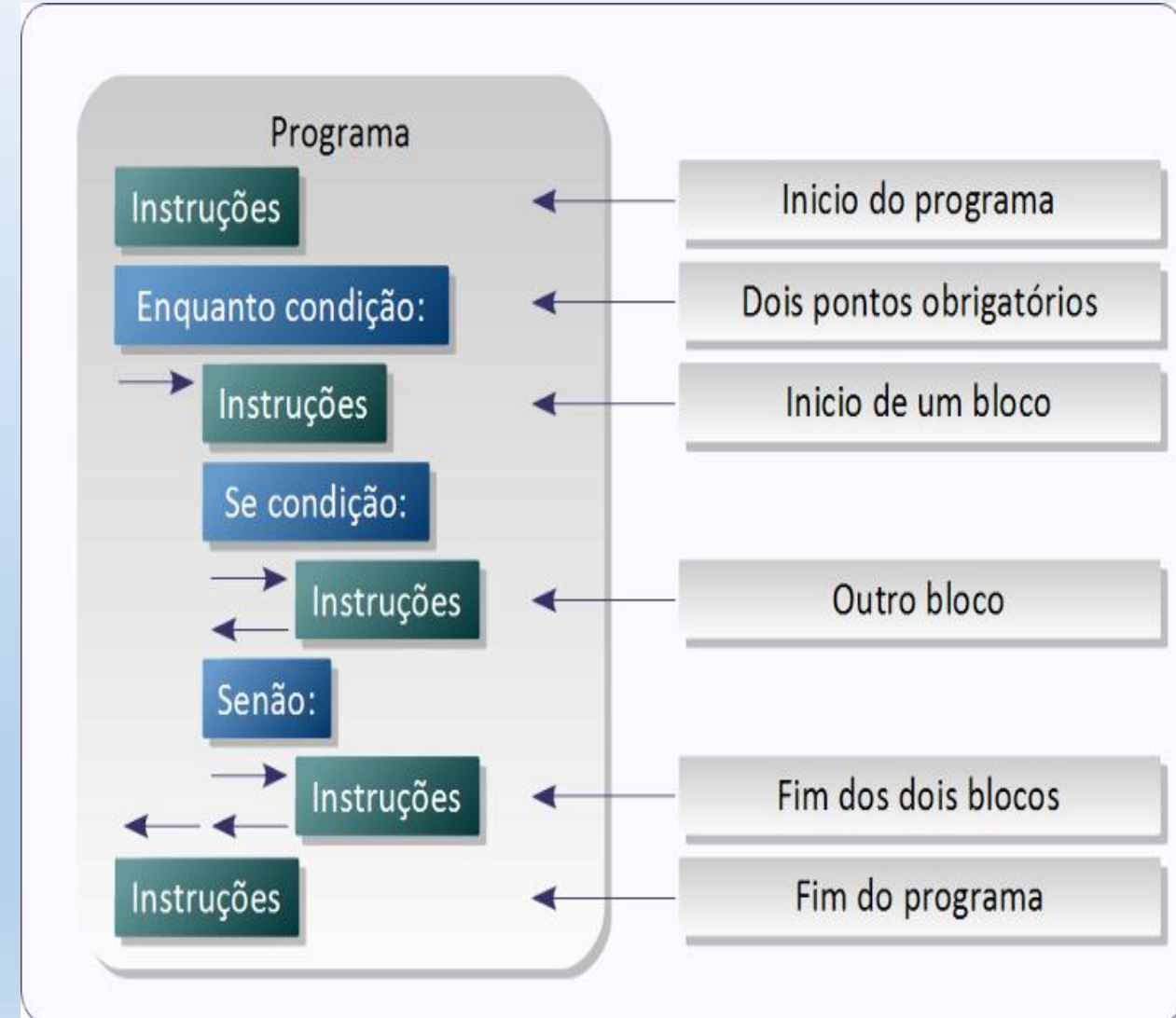
- Por que deu erro?

```
>>> # Using a \ for continue the multi-line statement
>>> a = 1 + 2 + 3 \
... 4 + 5 + 6 + 7
File "<stdin>", line 2
  4 + 5 + 6 + 7
    ^
SyntaxError: invalid syntax
```



Indentação no Python (cont)

- A maioria das linguagens de programação como C, C ++, Java usa {} para definir um bloco de código
- Python usa indentação
 - : para linha cabeçalho
 - Um bloco de código (corpo de uma função, loop, etc.) começa com o recuo (indentado com 4 espaços) e termina com a primeira linha sem recuo (não indentada)



Indentação no Python (cont)

```
// C program
if (x==42) {
    printf("The Answer to the Ultimate Question of
Life, the Universe, and Everything\n");
} else {
    printf("Just a number!\n");
}
```

```
from math import sqrt
n = input("Maximum Number? ")
n = int(n)+1
for a in range(1,n):
    for b in range(a,n):
        c_square = a**2 + b**2
        c = int(sqrt(c_square))
        if ((c_square - c**2) == 0):
            print(a, b, c)
```

Block 1

Block 2

Block 3

Block 2, continuation

Block 1, continuation

```
>>> a= 0; n = 9
>>> for b in range(a,n):
...     c_square = a**2 + b**2
...     c = 35
File "<stdin>", line 3
    c = 35
    ^
```

IndentationError: unexpected indent

Comentários no Python

- Comentários são importantes na documentação de código
- Para se escrever comentários no Python
 - Usa-se # para comentários de uma linha (*inline*)
 - Usa-se `"""` ou `'''` para comentários em várias linhas (*multiline*)

```
# This is a comment  
print('Hello')  
# This is a long comment  
# and it extends  
# to multiple lines
```

```
"""This is also a  
perfect example of  
multi-line comments"""
```

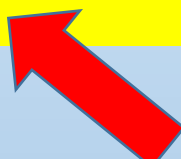
```
'''  
  
This is also a  
perfect example of  
multi-line comments  
'''
```

Identificadores

- Identificadores no Python são sensíveis ao caixa (*case sensitive*)
 - Temperature e temperature são identificadores diferentes

```
>>> temperature = 3
>>> Temperature = 4
>>> print (temperature,
Temperature)
3 4
```

```
>>> 9lives = 4
File "<stdin>", line 2
  9lives = 4
    ^
SyntaxError: invalid syntax
```



- Podem ser compostos por qualquer combinação de letras, números e outros caracteres (Unicode). Não podem começar com número e ter o caracter “-”

Identificadores

- Há cerca de 33 palavras reservadas

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

help> keywords

Here is a list of the Python keywords. Enter any keyword to get more help.

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

```
>>> None = 4
```

```
File "<stdin>", line 2
```

```
SyntaxError: can't assign to  
keyword
```

```
>>> try = 5
```

```
File "<stdin>", line 1
```

```
try = 5  
  ^
```

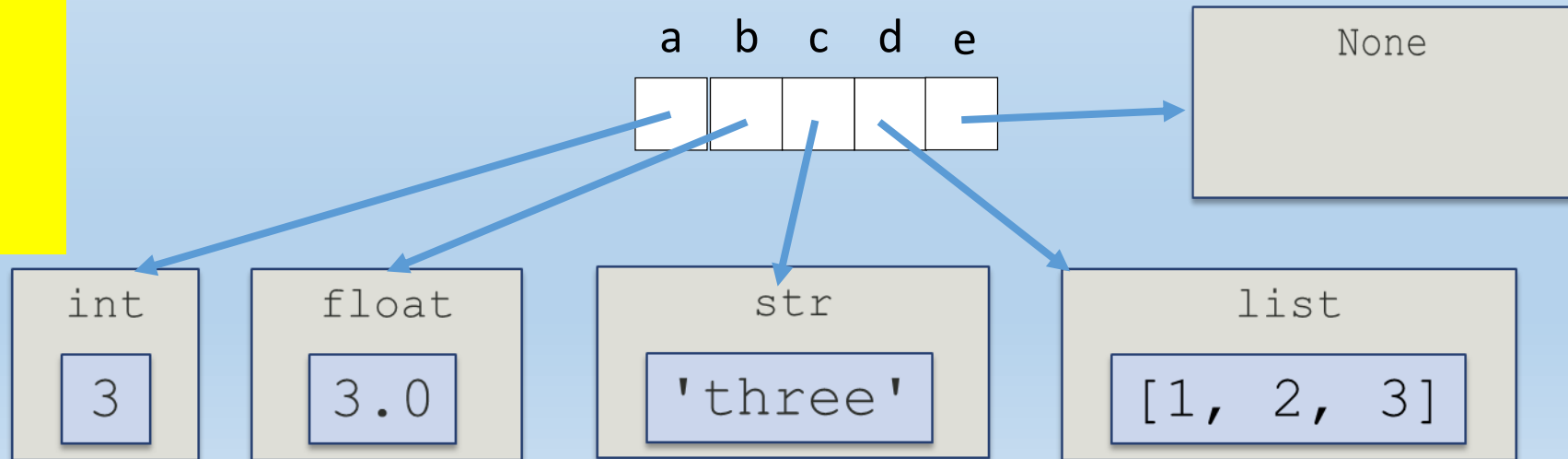
```
SyntaxError: invalid syntax
```



Objetos em Python

- A semântica dos identificadores no Python é similar a uma variável de referência no Java
 - Cada identificador é associado com um endereço de memória (*memory address*) do objeto do qual ele referencia
 - Um identificador pode ser associado a um objeto especial chamado None (similar a uma referência nula no Java)

```
>>> a=3  
>>> b=3.0  
>>> c='three'  
>>> d=[1,2,3]  
>>> e=None
```



Objetos em Python (cont)

- Diferente das linguagens Java ou C++, Python é uma linguagem tipada dinamicamente (*dynamically typed*)
 - Não há nenhuma instrução associando um identificador com um tipo de dado particular
 - Um identificador pode ser associado com qualquer tipo de objeto e mais tarde ele pode ser associado a outro objeto diferente do mesmo tipo
 - Ainda que o identificador não tenha um tipo definido, o objeto ao qual ele referencia tem um tipo definido

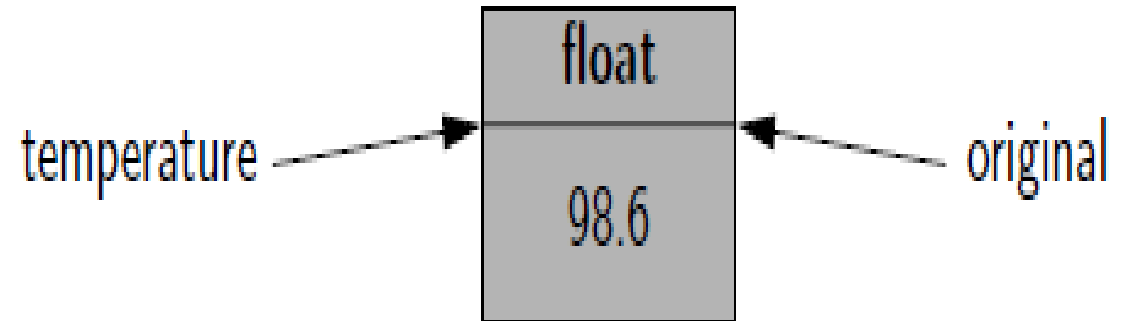
No exemplo ao lado, os caracteres 98.6 são identificados como um literal ponto flutuante. O identificador é então associado com uma instância da classe Float tendo aquele valor

```
>>> temperature = 98.6
>>> type (temperature)
<class 'float'>
```

Objetos em Python (cont)

- Pode-se estabelecer um *alias* assinalando um segundo identificador para o objeto existente

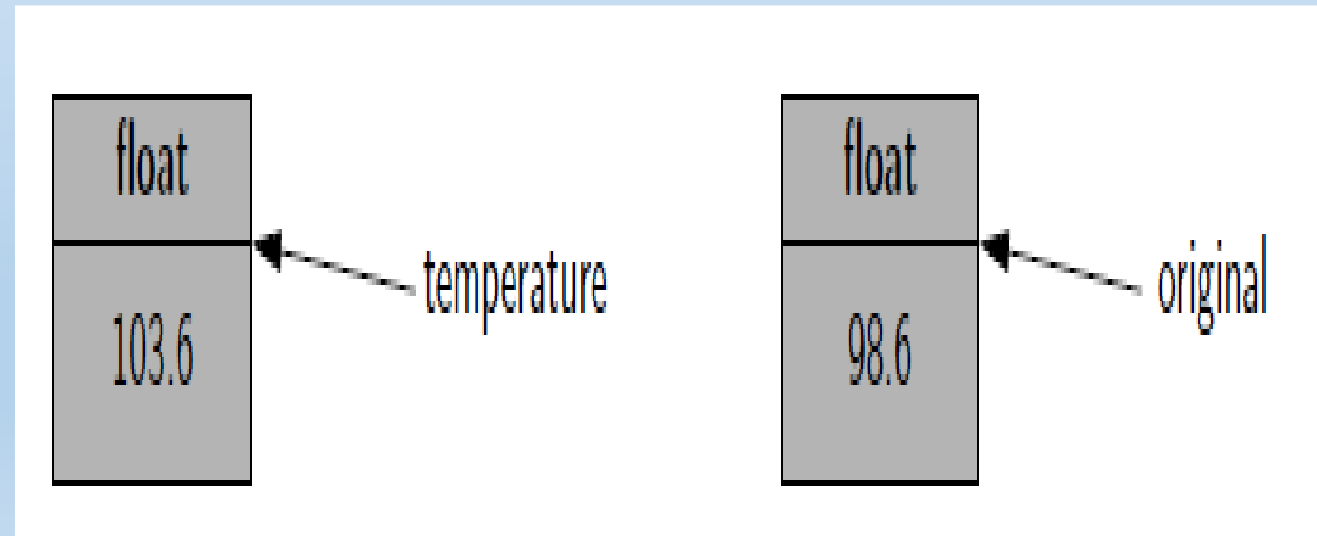
```
>>> temperature = 98.6  
>>> original = temperature  
>>> print (temperature, original)  
98.6 98.6
```



Objetos em Python (cont)

- E com a nova instrução a seguir
`temperature = temperature + 5`
- O que acontece?

```
>>> temperature = 98.6  
>>> original = temperature  
>>> print (temperature, original)  
98.6 98.6  
>>> temperature = temperature + 5  
>>> print (temperature, original)  
103.6 98.6
```



Objetos em Python (cont)

- O que acontece com os exemplos ao lado?

Python will also allocate the same memory address in the following two scenarios.

The strings don't have whitespaces and contain less than 20 characters.

In case of Integers ranging between -5 to +255.

```
>>> test1 = "Learn Python"
>>> id(test1)
2493823957744
>>> test2 = "Learn Python"
>>> id(test2)
2493824010672
>>> test3 = "Learn"
>>> id(test3)
2493824005040
>>> test4 = "Learn"
>>> id(test4)
2493824005040
```

```
>>> test5 = 200
>>> id(test5)
1479333104
>>> test6 = 200
>>> id(test6)
1479333104
>>> test7 = 300
>>> id(test7)
2493790266480
>>> test8 = 300
>>> id(test8)
2493823689616
```


Instanciação de Objetos em Python

- O processo de criação de uma nova instância de uma classe é conhecido como instanciação (*instantiation*) e a sintaxe para se instanciar um objeto envolve a invocação do construtor da classe
- Exemplo sem e com parametrização:

```
w = Widget()
```

```
z = Widget(a,b,c)
```

- Diversas classes pré-construídas (*built-in classes*) do Python suportam o que se chama de forma literal de criação de instância
- Exemplo:

```
temperature = 98.6
```

A instrução ao lado resulta na criação de uma nova instância da classe `Float` e o termo `98.6` está no formato literal

Chamada de Métodos em Python

- Python suporta funções tradicionais com passagem de parâmetros, como por exemplo:

`sorted(data)`

- As classes no Python podem definir um ou mais métodos (*member functions*) que são invocados sobre uma instância específica de uma classe por meio do operador “.”, como por exemplo, a classe List que possui um método chamado sort que pode ser invocado com a sintaxe a seguir:

`data.sort()`

Chamada de Métodos em Python

- A expressão do lado esquerdo do operador “.” identifica o objeto sobre o qual o método foi invocado, que frequentemente é um identificador
- Mas o operador “.” pode ser usado para invocar um método sobre o resultado imediato de alguma operação
- Exemplo:
 - Se `response` identifica uma instância de uma string contendo “YYY”, a instrução `response.lower().startswith('y')` primeiro avalia a chamada do método, `response.lower()`, que retorna uma nova instância da string, e então o método `startswith('y')` é chamado sobre a string intermediária

Chamada de Métodos em Python

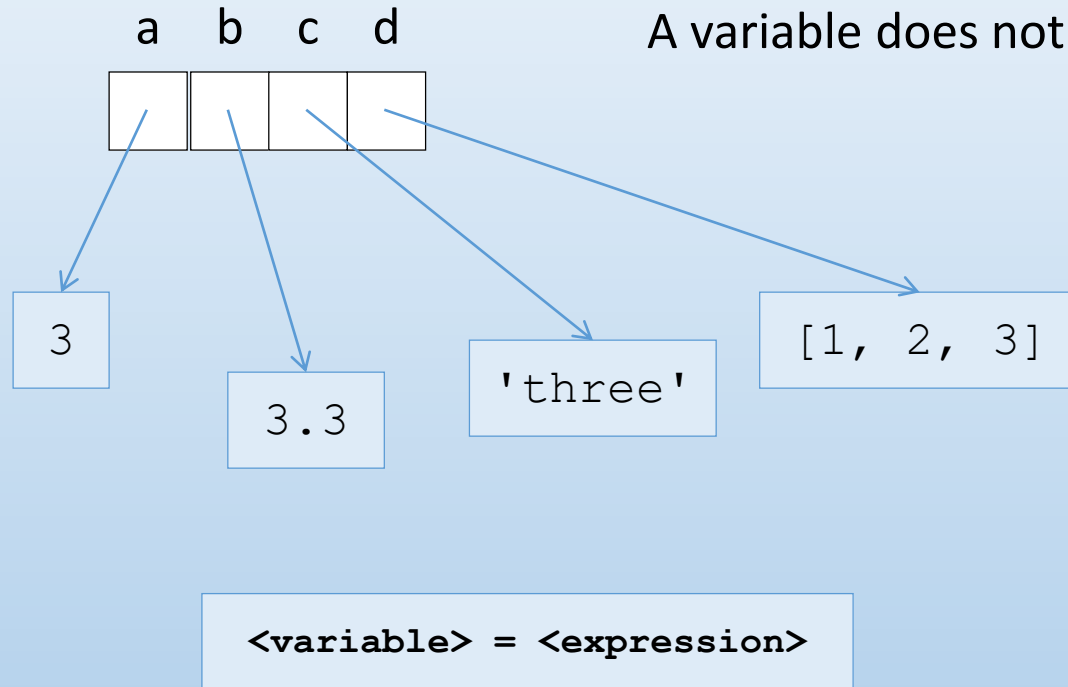
- Métodos que retornam informação sobre o objeto são chamados de métodos de acesso (*accessors methods*) e métodos que alteram o estado são chamados de métodos de atualização (*mutator ou update methods*)

```
>>> a = [1,2,3,4,5]
>>> a.count(1) # accessor method
1
>>> a.index(2) # accessor method
1
>>> a
[1, 2, 3, 4, 5]
>>> a.append(6) # mutator method
>>> a
[1, 2, 3, 4, 5, 6]
```

Classes Pré-Construídas em Python

- O Python disponibiliza de uma série de classes pré-construídas (*built-in classes*) e elas podem ser mutáveis ou imutáveis
 - Uma classe é imutável se todo objeto instanciado da classe possui um valor fixo que não pode ser alterado depois da instanciação do objeto
 - Uma classe é mutável se as informações armazenadas pelo objeto instanciado podem ser alteradas
- Exemplo:
 - A classe Float é imutável, pois uma vez que uma instância da classe tenha sido criada, seu valor não pode ser alterado

Classes Pré-Construídas em Python (cont.)

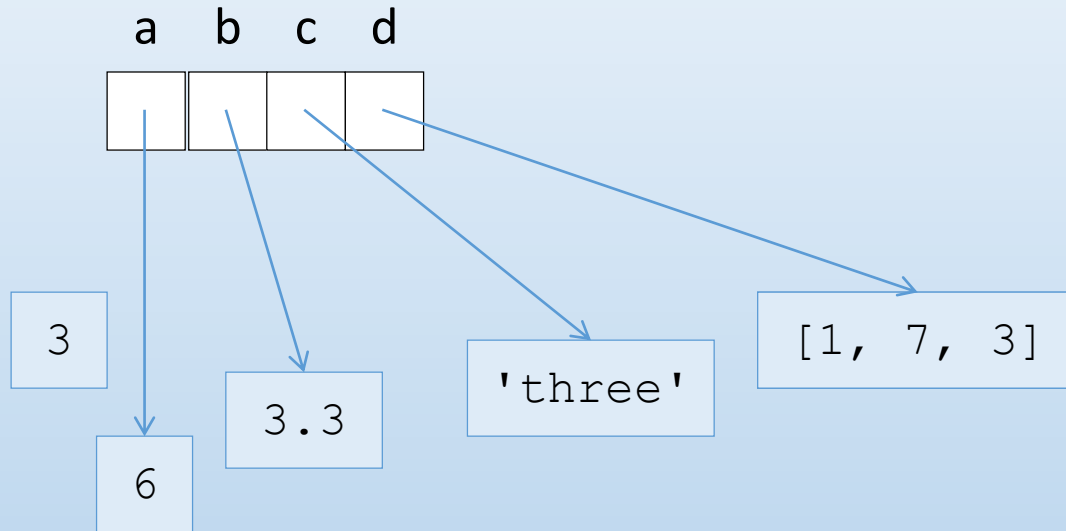


1. `<expression>` is evaluated and its value put into an object of appropriate type
2. The object is assigned name `<variable>`

```
>>> a
Traceback (most recent call
last):
  File "<pyshell#66>", line
  1, in <module>
    a
NameError: name 'a' is not
defined
>>> a = 3
>>> b = 2 + 1.3
>>> c = 'three'
>>> d = [1, 2] + [3]
```



Classes Pré-Construídas em Python (cont.)



The object (3) referred to by variable `a` does not change; instead, `a` refers to a new object (6)

- Integers are **immutable**

The object (`[1, 2, 3]`) referred to by `d` changes

- Lists are **mutable**

```
>>> a
3
>>> a = 6
>>> a
6
>>> d
[1, 2, 3]
>>> d[1] = 7
>>> d
[1, 7, 3]
```

Classes Pré-Construídas em Python (cont.)

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

As classes pré-fabricadas (*built-in classes*) podem ser criadas no formato literal ou com a utilização de construtores

Classe bool em Python

- A classe bool é usada para valores booleanos e as únicas duas instâncias da classe são expressas com literais:

True e False

- O construtor padrão (*default*) da classe retorna a instância False
- O Python permite a criação de valores booleanos a partir de um tipo não booleano usando a sintaxe bool(foo) para o valor foo. A interpretação depende do tipo do parâmetro:
 - Números são avaliados serem False se iguais a zero e True caso contrário; e
 - Strings e listas são avaliados serem False se vazios e True caso contrário

Classe bool em Python (cont.)

- Exemplo:

```
>>> z = ""
>>> s = '1'
>>> print (bool(z), bool(s))
False True
>>> a = 1
>>> b = 0
>>> print (bool(a), bool(b))
True False
```

```
>>> # Returns False as x is False
>>> x = False
>>> print(bool(x))
False
>>>
>>> # Returns False as x is not equal to y
>>> x = 5
>>> y = 10
>>> print(bool(x==y))
False
>>>
>>> # Returns False as x is an empty sequence
>>> x = ()
>>> print(bool(x))
False
```

Classe int em Python

- As classes `int` e `float` são utilizadas para representar os tipos numéricos primários no Python
- A classe `int` é projetada para representar valores inteiros com magnitude arbitrária (o limite é a memória)
 - Python automaticamente ajusta a representação interna do valor de acordo com sua magnitude
- O construtor da classe `int()`, retorna valor zero por padrão. Ele também pode ser usado para gerar um valor inteiro baseado em um valor existente de outro tipo
 - Se `f` representa um valor em ponto flutuante, `int(3.14)` produz o valor de 3
 - Se `137` representa o valor literal da string `'137'`, então `int(137)` produz 137

Classe int em Python (cont.)

- Exemplo:

```
>>> a = 0          # literal integer
>>> b = 137
>>> c = -23
>>> print (a,b,c)
0 137 -23
>>> d = 0b1011    # literal binary
>>> e = 0o52       # literal octal
>>> f = 0x7f       # literal hexadecimal
>>> print (d,e,f)
11 42 127
```

```
>>> int (3.14) # float to integer
3
>>> int (-3.9)
-3
>>> int ('137') # numeric string to integer
137
>>> int ('7f', 16) # conversion for another base than 10
127
>>> int ('hello') #string to integer
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hello'
```



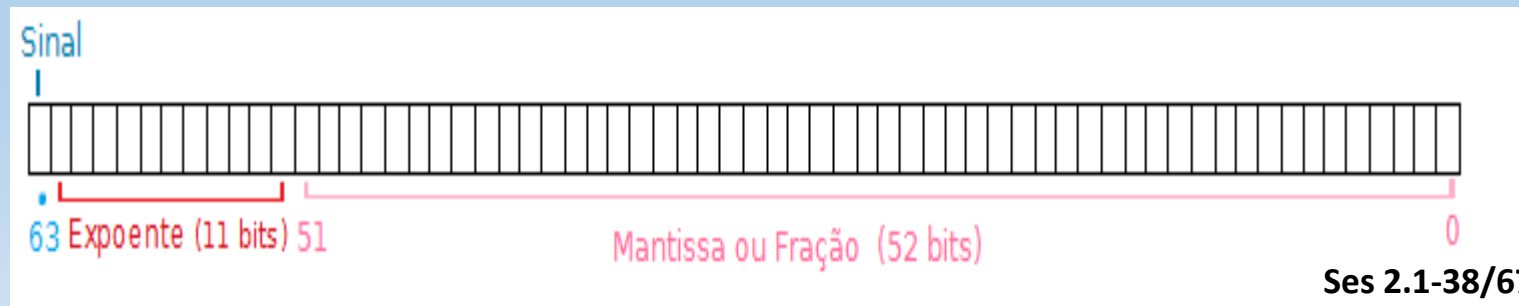
Classe int em Python (cont.)

- Exemplo:

```
>>> # Example 19
>>> 2**1030 # a very big number
1150523606311882180946755322104975829515
5052665230762065499525194094891251552061
6404933425486340823053516872883117691748
9016254520412266383954223556092634396017
7145179191391057683342054479750023644997
3089334403692056416436699176244143304390
7461294317845443268381352308735662374006
2701843509462805095950344781824
```


Classe float em Python

- A classe float é o tipo de dado com ponto flutuante no Python (com precisão similar ao double do Java)
 - Utiliza o formato IEEE 754 (dupla precisão em ponto flutuante), utilizando 64 bits para armazenamento dividido em três seções: sinal, expoente e mantissa
- O construtor da classe float(), retorna valor zero por padrão
- Quando fornecido um valor por parametrização, o construtor retorna o valor do parâmetro em ponto flutuante
 - float(2) retorna o valor em ponto flutuante 2.0
 - float('3.14') retorna 3.14



Classe float em Python (cont.)

- Exemplo:

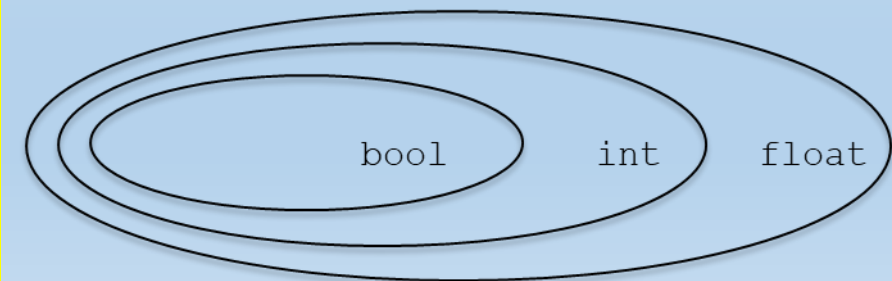
```
>>> # for integers
>>> print(float(10))
10.0
>>>
>>> # for floats
... print(float(11.22))
11.22
>>>
>>> # for string floats
... print(float("-13.33"))
-13.33
>>>
>>> # string float error
... print(float("abc"))
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ValueError: could not convert string to float: 'abc'
```



```
>>> 3 * 0.1 - 0.3
5.551115123125783e-17
>>> 0.3 - 0.1
0.19999999999999998
```



O que aconteceu?

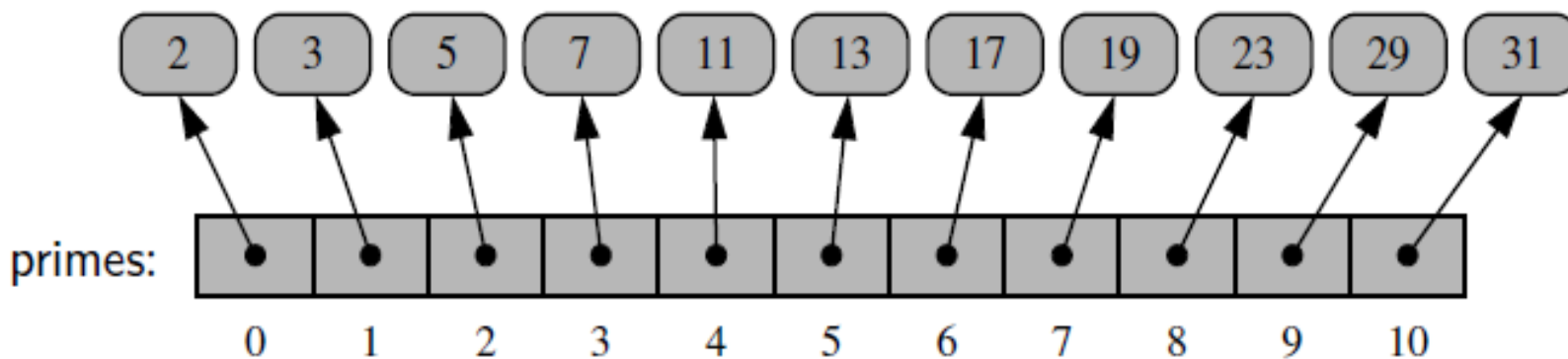


Tipos Sequências em Python

- As classes `list`, `tuple` e `str` são classes que representam tipos sequenciais (*sequence types*) no Python, representando uma coleção de valores em que a ordem é significativa
- A classe `list` é a mais geral, representando uma sequência de objetos quaisquer (similar ao *array* em outras linguagens)
- A classe `tuple` representa uma versão imutável da classe `list`
- A classe `str` é especialmente projetada para representar sequências imutáveis de caracteres de texto
 - O Python não disponibiliza uma classe para representar um caractere
 - São strings com tamanho 1

Classe `list` em Python

- Uma instância da classe `list` armazena uma sequência de objetos, isto é, uma sequência de referências para objetos na lista
 - Elementos de um objeto do tipo `list` podem ser de qualquer tipo (inclusive o objeto `None`)
 - Objetos do tipo `list` são sequências baseadas em *array* e são indexadas, sendo que um objeto `list` com comprimento n tem seus elementos indexados de 0 a $n-1$



Classe list em Python (cont.)

- Os caracteres [] são utilizados como delimitadores para objetos list compostos de literais
 - [] representa um objeto lista vazio
 - ['red', 'green', 'blue'] representa um objeto lista contendo três instâncias de strings
- O construtor padrão da classe list retorna um objeto do tipo list vazio
- O construtor da classe list aceita qualquer parâmetro que seja do tipo iterável (strings, list, tuples, etc.).
 - list('hello') produz um objeto do tipo list com caracteres individuais, ['h', 'e', 'l', 'l', 'o']
- Objetos do tipo list podem expandir e contrair suas capacidades de acordo com a necessidade

Classe list em Python (cont.)

- Exemplo:

```
>>> [] # empty list
[]
>>> a = ['red','green','blue'] # list with literals
>>> a
['red', 'green', 'blue']
>>> b = list() # empty list created by constructor
>>> b [1]
[]
>>> c = list('hello') # list created by constructor
>>> c
['h', 'e', 'l', 'l', 'o']
>>> d = [1,2,3,4]
>>> d.append(5) # element included in list
>>> d
[1, 2, 3, 4, 5]
```

Classe list em Python (cont.)

- Exemplo:

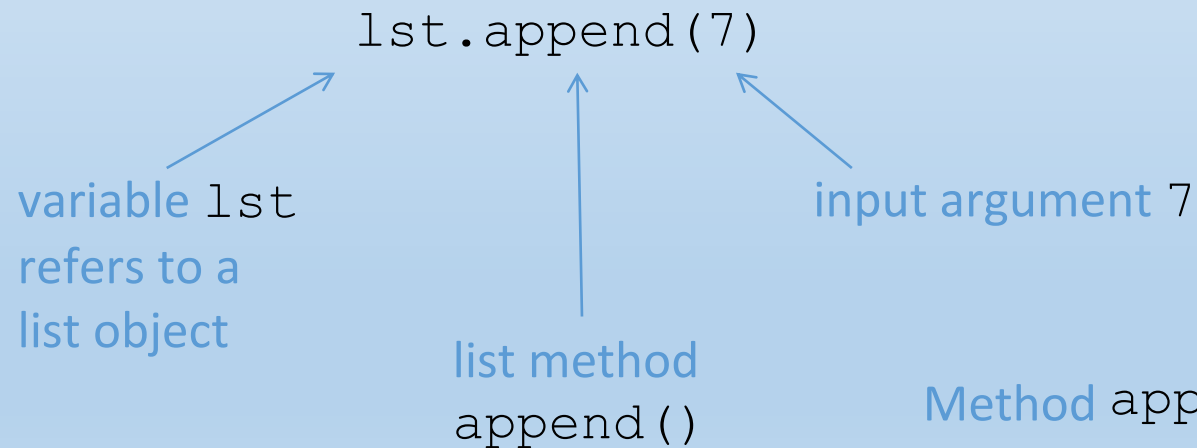
```
>>> x = [1, 2]      # list created using literals
>>> id(x)           # memory address of list
2129595828296
>>> x.append(4)     # element included in the list
>>> x
[1, 2, 4]
>>> id(x)           # memory address of list
2129595828296
```



A lista é um objeto mutável

Classe list em Python (cont.)

There are also functions that are called **on a list**;
such functions are called **list methods**



```
>>> lst = [1, 2, 3]
>>> len(lst)
3
>>> sum(lst)
6
>>> lst.append(7)
>>> lst
[1, 2, 3, 7]
>>>
```

Method `append()` **can't be called independently**; it must be called on some list object

Classe list em Python (cont.)

Usage	Explanation
<code>lst.append(item)</code>	adds item to the end of lst
<code>lst.count(item)</code>	returns the number of times item occurs in lst
<code>lst.index(item)</code>	Returns index of (first occurrence of) item in lst
<code>lst.pop()</code>	Removes and returns the last item in lst
<code>lst.remove(item)</code>	Removes (the first occurrence of) item from lst
<code>lst.reverse()</code>	Reverses the order of items in lst
<code>lst.sort()</code>	Sorts the items of lst in increasing order

Methods `append()`, `remove()`, `reverse()`, and `sort()` do not return any value; they, along with method `pop()`, modify list `lst`

```
>>> lst = [1, 2, 3]
>>> lst.append(7)
>>> lst.append(3)
>>> lst
[1, 2, 3, 7, 3]
>>> lst.count(3)
2
>>> lst.remove(2)
>>> lst
[1, 3, 7, 3]
>>> lst.reverse()
>>> lst
[3, 7, 3, 1]
>>> lst.index(3)
0
>>> lst.sort()
>>> lst
[1, 3, 3, 7]
>>> lst.remove(3)
>>> lst
[1, 3, 7]
>>> lst.pop()
7
>>> lst
[1, 3]
```

Classe `tuple` em Python

- A classe `tuple` é uma versão imutável de uma sequência, o que permite que suas instâncias tenham uma representação interna mais otimizada que objetos do tipo `list`
- Parênteses são utilizados para delimitar um objeto do tipo `tuple`
 - Um objeto do tipo `tuple` vazio é representado por `()`
- Para representar um objeto do tipo `tuple` de comprimento 1 como um literal, uma vírgula deve ser colocada depois do elemento, mas dentro do parêntesis
 - `(17,)` é um objeto do tipo `tuple` com um elemento

```
>>> a = (12)
>>> a
12
>>> type(a)
<class 'int'>
```

```
>>> a = (12,)
>>> a
(12,)
>>> type(a)
<class 'tuple'>
```

Classe tuple em Python (cont.)

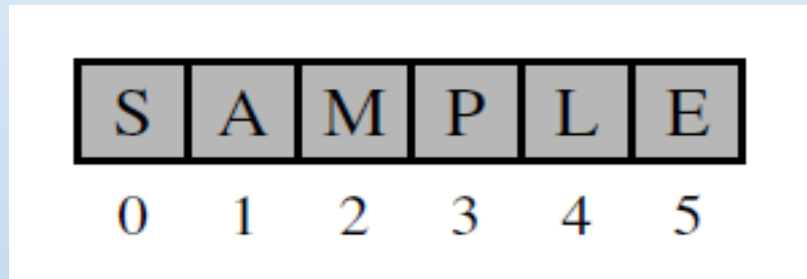
- Exemplo:

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
>>> u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
>>> t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
>>> v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```



Classe str em Python

- A classe str do Python foi especialmente projetada para a representação eficiente de uma sequência imutável de caracteres, usando o conjunto de caracteres Unicode



- O Python, de modo diferente de outras linguagens não tem um tipo char, de modo que um único caractere é representado por uma string de tamanho 1
- Há diversos modos de se representar uma string

Classe str em Python (cont.)

- Criação de Strings
 - Strings podem ser criadas com aspas simples, aspas duplas ou triplas

```
>>> # 4 ways to make a string
... str1 = 'This is a string. We built it with single quotes.'
>>> str2 = "This is also a string, but built with double quotes."
>>> str3 = '''This is built using triple quotes,
... so it can span multiple lines.'''
>>> str4 = """This too
... is a multiline one
... built with triple double-quotes."""
>>> str1
'This is a string. We built it with single quotes.'
>>> str2
'This is also a string, but built with double quotes.'
>>> str3
'This is built using triple quotes,\n so it can span multiple lines.'
>>> str4
'This too\n... is a multiline one\n built with triple double-quotes.'
```

Classe str em Python (cont.)

- Acesso ao caracteres em strings
 - Pode-se acessar um único caractere de uma string por meio de indexação
 - Pode-se recuperar (*slicing*) uma subcadeia de caracteres de um string
 - Python permite indexação negativa

P	R	O	G	R	A	M	I	Z	
0	1	2	3	4	5	6	7	8	9
-9	-8	-7	-6	-5	-4	-3	-2	-1	

```
>>> s = "The trouble is you think you have time."
>>> s[0] # indexing at position 0, which is the first char
'T'
>>> s[5] # indexing at position 5, which is the sixth char
'r'
>>> s[:4] # slicing, we specify only the stop position
'The '
>>> s[4:] # slicing, we specify only the start position
'trouble is you think you have time.'
>>> s[2:14] # slicing, both start and stop positions
'e trouble is'
>>> s[2:14:3] # slicing, start, stop and step (every 3 chars)
'erb '
>>> s[:] # quick way of making a copy
'The trouble is you think you have time.'
```


Classe str em Python (cont.)

- Operações com strings
 - Pode-se realizar uma série de operações com strings
 - Concatenação
 - Pertinência
 - ...

```
>>> str1 = 'Hello'
>>> str2 = 'World!'
>>>
>>> # using + for concatenating strings
... print('str1 + str2 = ', str1 + str2)
str1 + str2 = HelloWorld!
>>>
>>> # using * for multiply strings
... print('str1 * 3 =', str1 * 3)
str1 * 3 = HelloHelloHello
```

```
>>> #s[begin: end: step]
>>> #s[begin], s[begin + 1 * step], ... s[begin + i * step] for all (begin + i * step) < end
>>> s = "Python under Linux is great"
>>> s[::3]
'Ph d n e'
```

```
>>> # Pertinent test for string
>>> 'a' in 'program'
True
>>> 'at' not in 'battle'
False
>>>
```

Classe set em Python

- A classe set (mutável) representa a noção matemática de conjunto
 - Coleção de elementos sem ordem e sem repetição
- A classe disponibiliza de uma forma altamente otimizada para checar pertinência de elementos
- Entretanto somente objetos imutáveis podem ser elementos de um objeto set
 - Somente inteiros, ponto flutuante e strings
- Podem ser realizadas operações algébricas de conjuntos
 - União, intersecção, etc

Classe set em Python (cont.)

- Criação de um set
 - Pode-se criar um objeto set colocando-se todos os itens dentro de chaves, separadas por virgulas
 - Pode-se usar o construtor set()




```
>>> # set of integers
>>> my_set = {1, 2, 3}
>>> print(my_set)
{1, 2, 3}
>>> # set of mixed datatypes
>>> my_set = {1.0, "Hello", (1, 2, 3)}
>>> print(my_set)
{1.0, 'Hello', (1, 2, 3)}
>>> # set cannot have mutable items and [3, 4] is a mutable list
>>> my_set = {1, 2, [3, 4]}
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError: unhashable type: 'list'
>>> # we can make set from a list
>>> my_set = set([1,2,3,2])
>>> print(my_set)
{1, 2, 3}
```

Classe set em Python (cont.)

- Alteração de um set
 - Conjuntos são mutáveis
 - Entretanto, como são desordenados, a indexação não tem sentido
 - Pode-se adicionar ou remover um elemento de um set

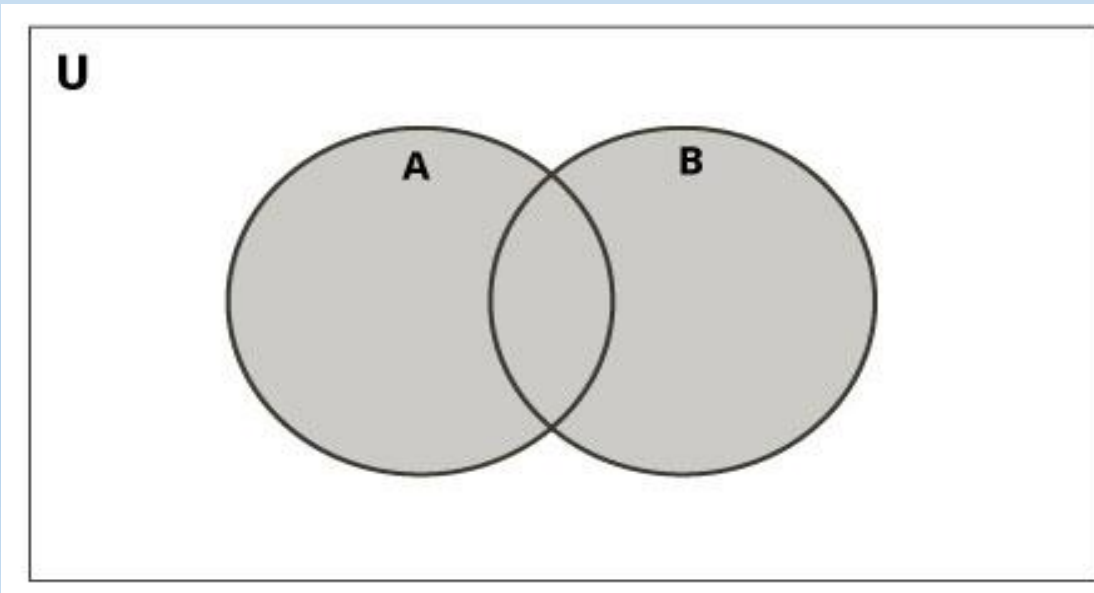
```
>>> # initialize my_set
>>> my_set = {1,3}
>>> print(my_set)
{1, 3}
>>> my_set[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
```



```
>>> # add an element
>>> my_set.add(2)
>>> print(my_set)
{1, 2, 3}
>>> # add list and set
>>> # Output: {1, 2, 3, 4, 5, 6, 8}
>>> my_set.update([4,5], {1,6,8})
>>> print(my_set)
{1, 2, 3, 4, 5, 6, 8}
>>> my_set.remove(6)
>>> print(my_set)
{1, 2, 3, 4, 5, 8}
```

Classe set em Python (cont.)

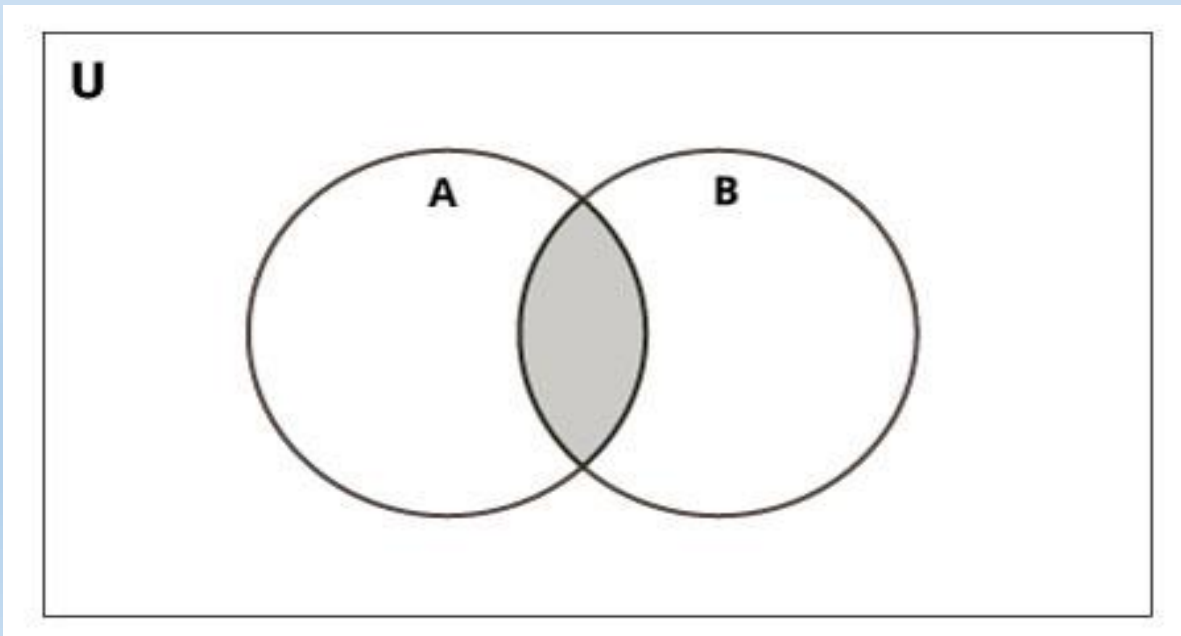
- Operações com conjuntos
 - Podem ser realizadas operações de álgebra relacional com sets
 - União



```
>>> # initialize A and B
... A = {1, 2, 3, 4, 5}
>>> B = {4, 5, 6, 7, 8}
>>>
>>> # use | operator
... # Output: {1, 2, 3, 4, 5, 6, 7, 8}
... print(A | B)
{1, 2, 3, 4, 5, 6, 7, 8}
```

Classe set em Python (cont.)

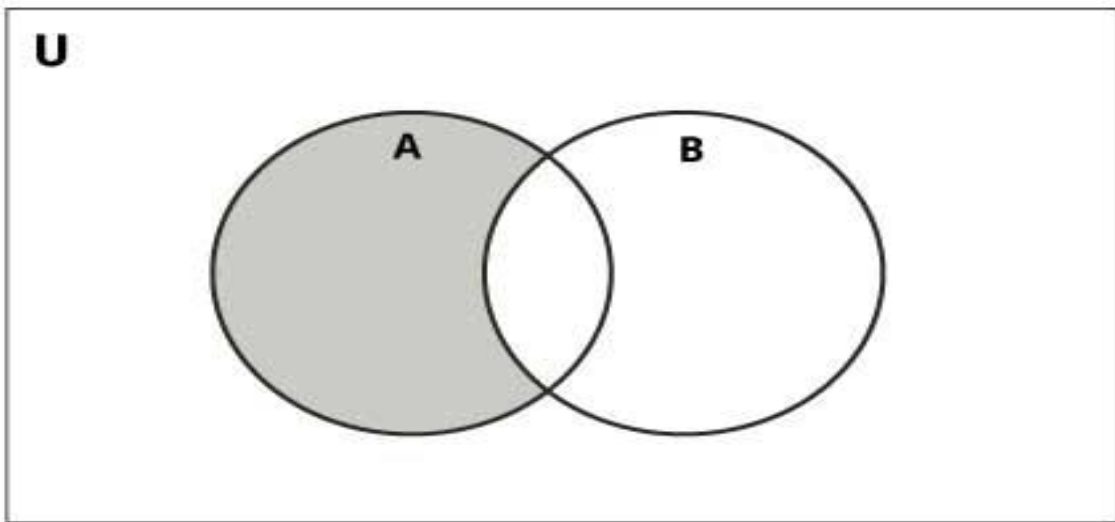
- Operações com conjuntos
 - Podem ser realizadas operações de álgebra relacional com sets
 - Intersecção



```
>>> # initialize A and B
... A = {1, 2, 3, 4, 5}
>>> B = {4, 5, 6, 7, 8}
>>>
>>> # use & operator
... # Output: {4, 5}
... print(A & B)
{4, 5}
```

Classe set em Python (cont.)

- Operações com conjuntos
 - Podem ser realizadas operações de álgebra relacional com sets
 - Diferença



```
>>> # initialize A and B
... A = {1, 2, 3, 4, 5}
>>> B = {4, 5, 6, 7, 8}
>>>
>>> # use - operator on A
... # Output: {1, 2, 3}
... print(A - B)
{1, 2, 3}
```


Classe dict em Python

- A classe dict representa um dicionário (ou mapeamento) de um conjunto de chaves distintas para valores
 - Por exemplo, um dicionário pode mapear um número de id de um estudante para valores adicionais contendo informações sobre o nome, endereço e matérias cursadas
- O dicionário é uma coleção não ordenada de itens com um par chave:valor
- Eles são otimizados para a recuperação de elementos quando se conhece o valor da chave

Classe dict em Python

- Criação de um dicionário
 - Cria-se um dicionário com um par de {} com os itens do dicionário separados por ,
 - Um elemento do dicionário tem um chave e valores expresso por um par chave: valor(es)
 - As chaves devem ser de tipos imutáveis (string, número ou tupla com elementos imutáveis) e devem ser únicas
 - Os valores podem ser de qualquer tipo e ser repetidos

```
>>> # empty dictionary
>>> my_dict = {}
>>> my_dict
{}
>>> # dictionary with integer keys
>>> my_dict = {1: 'apple', 2: 'ball'}
>>> my_dict
{1: 'apple', 2: 'ball'}
>>> # dictionary with mixed keys
>>> my_dict = {'name': 'John', 1: [2, 4, 3]}
>>> my_dict
{'name': 'John', 1: [2, 4, 3]}
>>> # using dict()
>>> my_dict = dict({1:'apple', 2:'ball'})
>>> my_dict
{1: 'apple', 2: 'ball'}
>>> # from sequence having each item as a pair
>>> my_dict = dict([(1,'apple'), (2,'ball')])
>>> my_dict
{1: 'apple', 2: 'ball'}
```

Classe dict em Python

- Acesso ao dicionário
 - Para se acessar elementos de um dicionário, são utilizadas as chaves
 - Chaves podem ser utilizadas dentro do colchetes, retornando `KeyError` em caso de não se encontrar a chave
 - Pode-se acessar utilizar também o método `get()`, retornando `None` no caso de não se encontrar a chave



```
>>> my_dict = {'name': 'Jack', 'age': 26}
>>> # Output: Jack
>>> print(my_dict['name'])
Jack
>>> # Output: 26
>>> print(my_dict.get('age'))
26
>>> # Trying to access keys which doesn't exist throws
error
>>> get_key = 0
>>> get_key = my_dict.get('address')
>>> print (get_key)
None
>>> my_dict['address']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'address'
```

Classe dict em Python

- Alteração no dicionário
 - Dicionários são estrutura de dados mutáveis
 - Pode-se adicionar novos elementos ou alterar o valor de elementos existentes no dicionário

```
>>> my_dict = {'name': 'Jack', 'age': 26}
>>> print(my_dict)
{'name': 'Jack', 'age': 26}
>>> # update value
>>> my_dict['age'] = 27
>>> #Output: {'age': 27, 'name': 'Jack'}
>>> print(my_dict)
{'name': 'Jack', 'age': 27}
>>> # add item
>>> my_dict['address'] = 'Downtown'
>>> # Output: {'address': 'Downtown', 'age': 27,
'name': 'Jack'}
>>> print(my_dict)
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

Exercise

Translate the following into Python algebraic or Boolean expressions and then evaluate them:

- a) The difference between Annie's age (25) and Ellie's (21)
- b) The total of \$14.99, \$27.95, and \$19.83
- c) The area of a rectangle of length 20 and width 15
- d) 2 to the 10th power
- e) The minimum of 3, 1, 8, -2, 5, -3, and 0
- f) 3 equals 4-2
- g) The value of 17//5 is 3
- h) The value of 17%5 is 3
- i) 284 is even
- j) 284 is even and 284 is divisible by 3
- k) 284 is even or 284 is divisible by 3

Exercise (cont.)

Write Python expressions involving strings `s1`, `s2`, and `s3` that correspond to:

- a) `'11'` appears in `s3`
- b) the blank space does not appear in `s1`
- c) the concatenation of `s1`, `s2`, and `s3`
- d) the blank space appears in the concatenation of `s1`, `s2`, and `s3`
- e) the concatenation of 10 copies of `s3`
- f) the total number of characters in the concatenation of `s1`, `s2`, and `s3`

Exercise (cont.)

String `s` is defined to be

```
'abcdefgh'
```

Write expressions using `s` and the indexing operator `[]` that return the following strings:

- a) `'a'`
- b) `'c'`
- c) `'h'`
- d) `'f'`

Exercise (cont.)

List `lst` is a list of prices for a pair of boots at different online retailers

- a) You found another retailer selling the boots for \$160.00; add this price to list `lst`
- b) Compute the number of retailers selling the boots for \$160.00
- c) Find the minimum price in `lst`
- d) Using c), find the index of the minimum price in list `lst`
- e) Using c) remove the minimum price from list `lst`
- f) Sort list `lst` in increasing order

Exercise (cont.)

Write a Python expression that assigns to variable `c`

- a) The length of the hypotenuse in a right triangle whose other two sides have lengths 3 and 4
- b) The value of the Boolean expression that evaluates whether the length of the above hypotenuse is 5
- c) The area of a disk of radius 10
- d) The value of the Boolean expression that checks whether a point with coordinates (5, 5) is inside a circle with center (0,0) and radius 7.