LABORATORIO DI:

METODI E MODELLI MATEMATICI IN PYTHON

A CURA DI: ANTONIO MIRARCHI & GIUSEPPE TROTTA

https://www.labmetodiemodelli.it/

Let’s Code!

https://www.labmetodiemodelli.it/

Dove eravamo rimasti?

I Tipi di Dato

https://www.labmetodiemodelli.it/

Basic Data Types

1. Numbers
2. Booleans
3. Strings
4. Containers (lists, dictionaries, sets, and tuples)

1.4 Container

Le liste

01 Oggetti Iterabili, che rappresentano dei contenitori di lunghezza arbitraria

02 Sequenze di Oggetti Eterogenei

03 Sequenze di oggetti Mutabili

https://www.labmetodiemodelli.it/

1.4.1 Lists

A list is the Python equivalent of an array, but is resizeable and can contain elements of different types:

```
xs = [3, 1, 2] # Create a list
print(xs, xs[2]) # Prints "[3, 1, 2] 2"
print(xs[-1]) # Negative indices count from the end of the list; prints "2"
xs[2] = 'foo' # Lists can contain elements of different types
print(xs) # Prints "[3, 1, 'foo']"
xs.append('bar') # Add a new element to the end of the list
print(xs) # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop() # Remove and return the last element of the list
print(x, xs) # Prints "bar [3, 1, 'foo']"
```

https://docs.python.org/3.5/tutorial/datastructures.html#more-on-lists

1.4.1 Lists

Slicing: In addition to accessing list elements one at a time, Python provides concise syntax to access sublists; this is known as slicing:

```
nums = list(range(5)) # range is a built-in function that creates a list of integers
print(nums) # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4]) # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]”
print(nums[2:]) # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2]) # Get a slice from the start to index 2 (exclusive); prints "[0, 1]”
print(nums[:]) # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1]) # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9] # Assign a new sublist to a slice
print(nums) # Prints "[0, 1, 8, 9, 4]"
```

https://docs.python.org/3.5/tutorial/datastructures.html#more-on-lists

1.4.1 Lists - Loops

You can loop over the elements of a list like this:

```
animals = ['cat', 'dog', 'monkey’]
for animal in animals:
      print(animal)

# Prints "cat", "dog", "monkey", each on its own line.
```

https://docs.python.org/3.5/tutorial/datastructures.html#more-on-lists

1.4.1 Lists - enumerate

If you want access to the index of each element within the body of a loop, use the built-in enumerate function:

```
animals = ['cat', 'dog', 'monkey’]
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))

# Prints "#1: cat", "#2: dog", "#3: monkey", each on its own line
```

https://docs.python.org/3.5/tutorial/datastructures.html#more-on-lists

1.4.1 Lists - example

Converts this code using the list comprehensions

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)       # Prints [0, 1, 4, 9, 16]
```

https://docs.python.org/3.5/tutorial/datastructures.html#more-on-lists

1.4.1 Lists - example

Converts this code using the list comprehensions

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)       # Prints [0, 1, 4, 9,
16]
```

https://docs.python.org/3.5/tutorial/datastructures.html#more-on-lists

1.4.1 Lists - conditions

List comprehensions can also contain conditions:

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares) # Prints "[0, 4, 16]"
```

https://docs.python.org/3.5/tutorial/datastructures.html#more-on-lists

Le Tuple

01 Oggetti Iterabili, che rappresentano dei contenitori di lunghezza arbitraria

02 Sequenze di Oggetti Eterogenei

03 Sequenze di oggetti Immutabili

https://www.labmetodiemodelli.it/

1.4.1 Tuples - Comprehensions

A tuple is an (immutable) ordered list of values. A tuple is in many ways similar to a list; one of the most important differences is that tuples can be used as keys in dictionaries and as elements of sets, while lists cannot.

```
t = (5, 6) # Create a tuple
print(type(t)) # Prints "<class 'tuple'>"
t2 = ("a", "b", "c", "d", "e")
t2[0] -> 'a'
t2[1] -> 'b'
t2[-1] -> 'e'
t2[1:3] -> ('b', 'c')
IMMUTABILI
t2['a'] = 15 -> TypeError: 'tuple' object does not support item assignment
dir(t2) -> lista dei metodi supportati dalle tuple
```

https://docs.python.org/3.5/tutorial/datastructures.html#tuples-and-sequences

I Dizionari

01 Oggetti Iterabili, che rappresentano dei contenitori di lunghezza arbitraria

02 Coppie chiave-valore

03 Sequenze di oggetti Mutabili ma non ordinabili

1.4 Container

	Mutable	Ordered	Indexing/Slicing	Duplicate Elem
Liste	✓	✓	✓	✓
Tuple	X	✓	✓	✓
Set	✓	X	X	X

https://www.labmetodiemodelli.it/

1.4.2 Dictionaries

A dictionary stores (key, value) pairs, similar to a Map in Java or an object in Javascript. You can use it like this:

```
d = {'cat': 'cute', 'dog': 'furry'} # Create a new dictionary with some data
print(d['cat']) # Get an entry from a dictionary; prints "cute"
print('cat' in d) # Check if a dictionary has a given key; prints "True"
d['fish'] = 'wet' # Set an entry in a dictionary
print(d['fish']) # Prints "wet"
print(d['monkey']) # KeyError: 'monkey' not a key of d
print(d.get('monkey', 'N/A')) # Get an element with a default; prints "N/A"
print(d.get('fish', 'N/A')) # Get an element with a default; prints "wet"
del d['fish'] # Remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

https://docs.python.org/3.5/library/stdtypes.html#dict

1.4.2 Dictionaries - Loops

It is easy to iterate over the keys in a dictionary:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))

# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

https://docs.python.org/3.5/library/stdtypes.html#dict

1.4.2 Dictionaries - Comprehensions

If you want access to keys and their corresponding values, use the items method:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))


# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

https://docs.python.org/3.5/library/stdtypes.html#dict

Let’s Code!

https://www.labmetodiemodelli.it/

I Set

01 Oggetti Iterabili, che rappresentano dei contenitori di lunghezza arbitraria

02 Non Ordinati e senza duplicati

03 Sequenze di oggetti Mutabili ma non ordinabili

04 Operazioni degli insiemi

https://www.labmetodiemodelli.it/

1.4.3 Sets

A set is an unordered collection of distinct elements. As a simple example, consider the following:

```
animals = {'cat', 'dog’}
print('cat' in animals) # Check if an element is in a set;
prints "True" print('fish' in animals) # prints "False"
animals.add('fish') # Add an element to a set
print('fish' in animals) # Prints "True"
print(len(animals)) # Number of elements in a set; prints "3"
animals.add('cat') # Adding an element that is already in the set does nothing
print(len(animals)) # Prints "3"
animals.remove('cat') # Remove an element from a set
print(len(animals)) # Prints "2"
```

https://docs.python.org/3.5/library/stdtypes.html#set

1.4.3 Sets - Loops

Iterating over a set has the same syntax as iterating over a list; however since sets are unordered, you cannot make assumptions about the order in which you visit the elements of the set:

```
animals = {'cat', 'dog', 'fish’}
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))


# Prints "#1: fish", "#2: dog", "#3: cat"
```

https://docs.python.org/3.5/library/stdtypes.html#set

1.4.3 Sets - Comprehensions

Like lists and dictionaries, we can easily construct sets using set comprehensions:

```
from math import sqrt

nums = {int(sqrt(x)) for x in range(30)}
print(nums)


# Prints "{0, 1, 2, 3, 4, 5}"
```

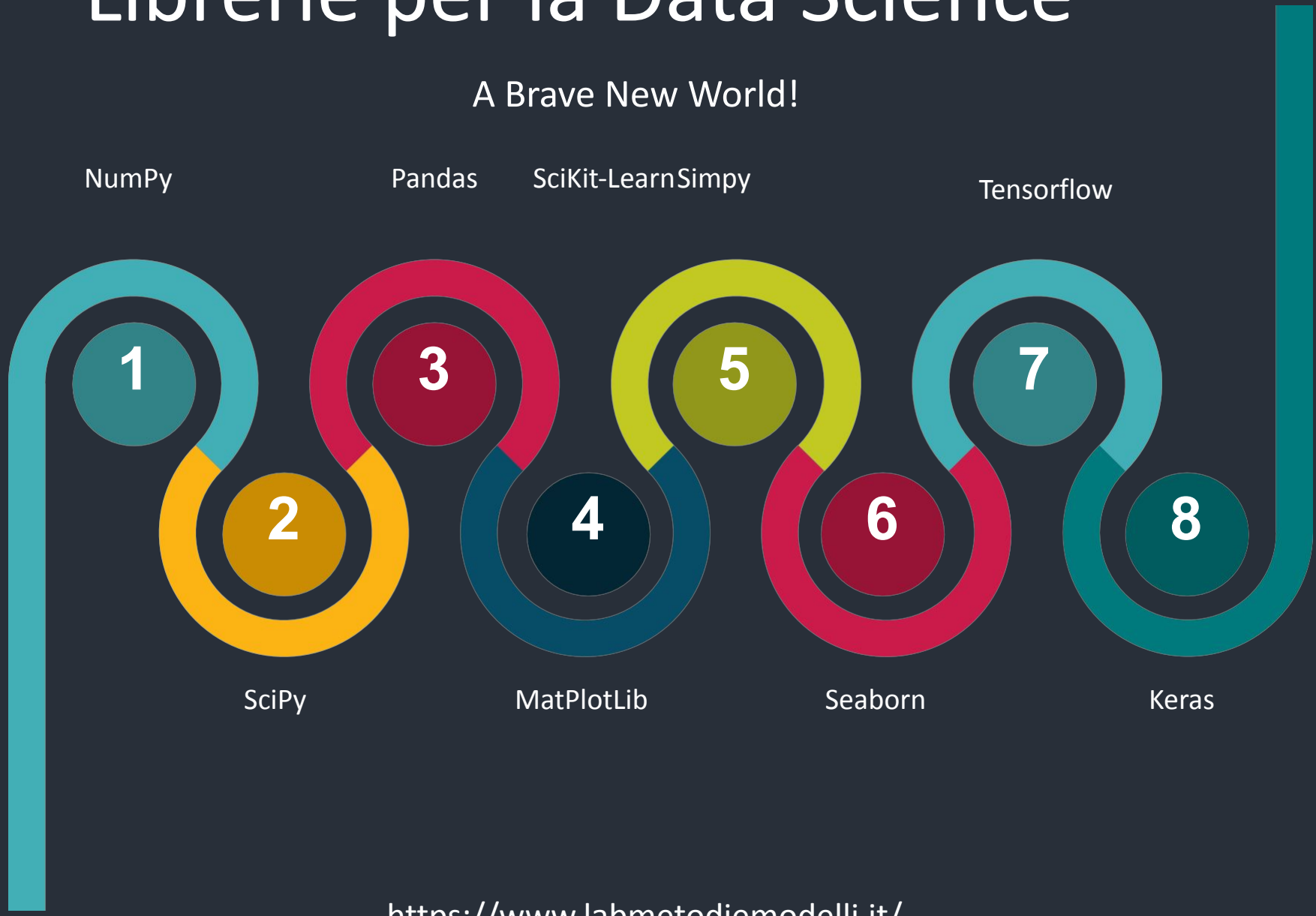
https://docs.python.org/3.5/library/stdtypes.html#set

Let’s Code!

https://www.labmetodiemodelli.it/

Librerie per la Data Science

A Brave New World!



https://www.labmetodiemodelli.it/