جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

المدرسة الوطنية العليا للفنون و المهن
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS

Python programming
- Data Structures -

EL Moukhtar ZEMMOURI

ENSAM – Meknès

Version 1.0 – 2019

Python Data Structures

Compound types

- Sequences
  - Immutable sequences
    - Strings **`str`**
    - Tuples **`tuple`**
    - Bytes **`byte`**
  - Mutable sequences
    - Lists **`list`**
    - Byte Arrays **`bytearray`**
- Set types
  - Sets **`set`**
  - Frozen sets **`frozenset`**
- Mappings
  - Dictionaries **`dict`**

Sequences

- Represent finite ordered sets indexed by non-negative numbers
- Built-in function **`len()`** returns the number of items
- A sequence `a` is indexed by `0` to `len(a)-1`
  - Item `i` of sequence `a` is accessed by `a[i]`
- Sequences support slicing
  - `a[i:j]` selects items with index *k*, *i* <= *k* < *j*.
- **Immutable** sequences
  - Objects cannot change once it is created
- **Mutable** sequences
  - Object can be changed after creation

Sequences

- Common operations :
  - **`x in s`** True if an item of *s* is equal to *x*, else False
  - **`x not in s`** False if an item of *s* is equal to *x*, else True
  - **`s + t`** the concatenation of *s* and *t*
  - **`s * n`** or **`n * s`** equivalent to adding *s* to itself *n* times
  - **`min(s)`** smallest item of *s*
  - **`max(s)`** largest item of *s*

Set types

- Represent unordered, finite sets of unique, immutable objects
  - ➔ Cannot be indexed
  - Can be iterated over
  - Built-in function **`len()`** returns the number of items
- Common uses for sets :
  - membership testing,
  - removing duplicates from a sequence,
  - computing mathematical operations such as intersection, union, difference, and symmetric difference.

Mappings

- Represent finite sets of objects indexed by arbitrary index sets
  - ➔ index may be string, numbers, ...
  - `a[k]` selects the item indexed by `k` from the mapping `a`
  - Built-in function **`len()`** returns the number of items

Strings

Strings

- Strings are arrays of bytes representing Unicode characters
- A Python str can be enclosed in single or double quotes:
  - `'Hello World'` is the same as `"Hello World"`.
  - String literals can span multiple lines using triple-quotes: `"""..."""` or `'''...'''`
- Python does not have a char data type.
  - ➔ a single character is a string with a length of 1.

Strings

- Declaration, indexing, slicing

```
>>> a = "Hello world!"
>>> b = '"Hello world!", I said'
>>> c = """ this is
... a multiline block
... of text """
```

```
>>> s = "Hello World!"
>>> s[0] = 'h'
TypeError: 'str' object does
not support item assignment
```

```
>>> a[0]
'H'
>>> b[0]
'"'
>>> a[11]
'!'
>>> a[12]
IndexError: string index out
of range
```

```
>>> a[-1]
'!'
>>> a[-len(a)]
'H'
>>> a[0]
'H'
>>> a[len(a)-1]
'!'
```

```
>>> b[1:6]
'Hello'
>>> a[6:]
'world!'
>>> a[:5]
'Hello'
>>> b[-6:]
'I said'
```

Strings

- Operators, interpolation, functions, and methods

```
>>> p = 98.1234
>>> r = 97.4321
>>> print("Precision = " + str(p) + "%")
Precision = 98.1234%
>>> s = "Precision %.2f et Recall %.2f" %(p, r)
>>> s
'Precision 98.12 et Recall 97.43'
```

```
>>> "Hello" + "World"
'HelloWorld'
>>> "Hello" * 3
'HelloHelloHello'
>>> 'lo' in "Hello World"
True
```

```
>>> s = "Hello Python Programmers !"
>>> s.upper()
'HELLO PYTHON PROGRAMMERS !'
>>> s.split(" ")
['Hello', 'Python', 'Programmers', '!']
>>> s.replace("Python", "C/C++")
'Hello C/C++ Programmers !'
>>> print("Precision={} Recall={}".format(p, r))
Precision=98.1234 Recall=97.4321
>>> help(str) #for more methods on strings
```

```
>>> a = "Hello world!"
>>> ord('A') # the Unicode of 'A'
65
>>> ord('ض')
1590
>>> chr(65)
'A'
>>> chr(8364)
'€'
```


Quick exercise

- Use the method `find` to print out indices of all occurrences of a substring `sub` in a string `s`.

Lists

Lists

- A list is a collection of items
  - Can contain items of different types
  - But usually the items all have the same type

```
>>> t = [10, 2, 25, 2]
>>> type(t)
<class 'list'>
>>> print(t)
[10, 2, 25, 2]
>>> t*2
[10, 2, 25, 2, 10, 2, 25, 2]
>>> t = t + ["Hello", 1.5]
>>> t
[10, 2, 25, 2, 'Hello', 1.5]
>>> a = [[1, 2], [10, 20]]
>>> a
[[1, 2], [10, 20]]
```

```
>>> len(t)
6
>>> t[0]
10
>>> t[4]
'Hello'
>>> t[-1]
1.5
>>> t[-6]
10
>>> t[1:4]
[2, 25, 2]
>>> t[4] = 100
>>> t
[10, 2, 25, 2, 100, 1.5]
```

```
>>> t = list(range(10))
>>> s = 0
>>> for e in t :
...     s += e
...
>>> print(s)
45
>>> sum(t)
45
```

Lists

- List methods / copying lists

```
>>> L = [8, 1, 5, 6, 4, 10]
>>> L.append(15)
>>> L
[8, 1, 5, 6, 4, 10, 15]
>>> L.insert(1, 25)
>>> L
[8, 25, 1, 5, 6, 4, 10, 15]
>>> L.pop()
15
>>> L
[8, 25, 1, 5, 6, 4, 10]
>>> L.pop(2)
1
>>> L
[8, 25, 5, 6, 4, 10]
```

```
>>> L.index(5)
2
>>> L.index(2)
ValueError: 2 is not in list
>>> 2 in L
False
>>> L.sort()
>>> L
[4, 5, 6, 8, 10, 25]
>>> L.reverse()
>>> L
[25, 10, 8, 6, 5, 4]
>>> L.clear()
>>> L
[]
```

```
>>> a = [1, 2, 3]
>>> b = a
>>> a[0] = 10
>>> a
[10, 2, 3]
>>> b
[10, 2, 3]
>>> a == b
True
>>> a is b
True
>>> c = a.copy()
>>> a[1] = 20
>>> a
[10, 20, 3]
>>> c
[10, 2, 3]
>>> a == c
False
```


Quick exercise

- A matrix (2d-array) can be represented in Python as a list of lists.
  - Create a matrix of zeros with size (n, m)
  - Create the identity matrix with size (n, n)

Lists

- List comprehensions

```
>>> a = [x**2 for x in range(1, 6)]
>>> a
[1, 4, 9, 16, 25]
>>> b = [x**2 for x in range(10) if x%2]
>>> b
[1, 9, 25, 49, 81]
```

```
>>> n = 3
>>> m = 4
>>> M = [[0 for j in range(m)] for i in range(n)]
>>> M
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```


Exercise

- A matrix (2d-array) can be represented in Python as a list of lists.
- Define a function that returns a random matrix with size (n, m) and values between 0.0 and 1.0
  - Hint : use function of random module
- Define a function that computes the sigmoid of a matrix.
- Define a function that computes the softmax of a matrix.

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$softmax(x) = softmax\begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \cdots & x_{mn} \end{bmatrix} = \begin{bmatrix} \frac{e^{x_{11}}}{\sum_j e^{x_{1j}}} & \frac{e^{x_{12}}}{\sum_j e^{x_{1j}}} & \frac{e^{x_{13}}}{\sum_j e^{x_{1j}}} & \cdots & \frac{e^{x_{1n}}}{\sum_j e^{x_{1j}}} \\ \frac{e^{x_{21}}}{\sum_j e^{x_{2j}}} & \frac{e^{x_{22}}}{\sum_j e^{x_{2j}}} & \frac{e^{x_{23}}}{\sum_j e^{x_{2j}}} & \cdots & \frac{e^{x_{2n}}}{\sum_j e^{x_{2j}}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{e^{x_{m1}}}{\sum_j e^{x_{mj}}} & \frac{e^{x_{m2}}}{\sum_j e^{x_{mj}}} & \frac{e^{x_{m3}}}{\sum_j e^{x_{mj}}} & \cdots & \frac{e^{x_{mn}}}{\sum_j e^{x_{mj}}} \end{bmatrix}$$

Dictionaries

Dictionaries

- A dictionary is an unordered collection of (key, value) pairs.
  - A dictionary stores data as a key:value pair
  - Dictionaries are indexed by *keys*

```
>>> d = {"fes":60, "rabat":120, "ifrane":55}
>>> d
{'rabat': 120, 'fes': 60, 'ifrane': 55}
>>> type(d)
<class 'dict'>
>>> d['rabat']
120
>>> d["casa"] = 300
>>> d
{'rabat': 120, 'casa': 300, 'fes': 60, 'ifrane': 55}
>>> d.pop('rabat')
120
>>> d
{'casa': 300, 'fes': 60, 'ifrane': 55}
```

```
>>> list(d)
['casa', 'fes', 'ifrane']
>>> 'fes' in d
True
>>> for k in d.keys():
...     print(k, d[k])
...
casa 300
fes 60
ifrane 55
>>> for k, v in d.items():
...     print(k, v)
...
casa 300
fes 60
ifrane 55
>>> {x:x**2 for x in (2, 4, 8)}
{8: 64, 2: 4, 4: 16}
```

Tuples

Tuples

- A tuple is an immutable list
- tuples are usually used :
  - to pass arguments into functions
  - to return values from functions
  - as keys in dictionaries.

```
>>> t = (100, 200, 'Hello', 12.5)
>>> type(t)
<class 'tuple'>
>>> t
(100, 200, 'Hello', 12.5)
>>> t[0]
100
>>> t[len(t)-1]
12.5
>>> t[1] = 10
TypeError: 'tuple' object does not support item
assignment
>>> a, b, c, d = t  # tuple unpacking
>>> c
'Hello'
>>> L = list(t)
>>> L
[100, 200, 'Hello', 12.5]
>>> pts = [(x, x*x) for x in range(5)]
>>> pts
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16)]
```