Outline

- Sequences
  - lists
  - strings
  - tuples
- Dictionaries
- Functions
Sequences
Sequences

- **Python sequence types:**
  - Strings  'Ali'
  - Lists    [110, 'me', 5]
  - Tuples   (3, 7)

- Sequences are **iterable**: you can traverse their items one at a time

```python
var my_list = ['ali', 20, 14]
for x in my_list:
    print(x)
```
Indexing and Slicing

- We can directly index items in a sequence, or slice a subsequence.
- Example: \(s = \text{"Python"}\)
  - \(s[0]\)  \(\text{\textquotesingle}P\text{\textquotesingle}\)
  - \(s[0:2]\)  \(\text{\textquotesingle}Py\text{\textquotesingle}\)
  - \(s[:2]\)  \(\text{\textquotesingle}Py\text{\textquotesingle}\)
  - \(s[2:]\)  \(\text{\textquotesingle}thon\text{\textquotesingle}\)
  - \(s[:]\)  \(\text{\textquotesingle}Python\text{\textquotesingle}\)
  - \(s[-1]\)  \(\text{\textquotesingle}n\text{\textquotesingle}\)
Sequence Operators

- **Operators**
  - `+`: concatenation
  - `*`: repetition/duplication
  - `in`: membership
  - `not in`: non-membership

- **Functions**
  - `len`
  - `max`, `min`, `sum`
  - `sorted`
Lists

• **list** is a Python type that acts most like other languages’ arrays

• Useful methods
  – append, extend, insert, remove, pop, index, count, sort, reverse

```
a = [20, 14]
a.append(10)  # [20, 14, 10]
a.sort()     # [10, 14, 20]
```

• We can use lists as **stacks** and **queues** (or alternatively, use **deque**)
List Comprehensions

• A **list comprehension** is a list defined by a "logic" that builds the list values/objects

```python
>>> evens = [x for x in range(10) if x % 2 == 0]
>>> evens
[0, 2, 4, 6, 8]
```

• A **generator expression** is almost the same, except it performs "lazy-evaluation" of objects

```python
>>> (x for x in range(1000) if x % 2 == 0)
<generator object <genexpr> at 0x02D32350>
```
Strings

- **Useful sting methods**
  - **count**: Number of occurrences of substring in string
  - **find**: Search for substring [also index, rfind, rindex]
  - **join**: Merge substrings into single delimited string
  - **replace**: Search and replace (sub)string
  - **split**: Split string into substrings [also splitlines]
  - **startswith**: Does string start with substring [also endswith]
  - **strip**: Remove whitespace around [also rstrip, lstrip]
  - **upper**: UPPERCASE string [also lower]
  - **isupper**: Is string all UPPERCASE? [also islower]
String Formatting

- Using `%` operator

```python
>>> 'Number of %s is %d' % ('steps', 100)
"Number of steps is 100"
```

- Using `format` method

```python
>>> 'Number of {0} is {1}'.format('steps', 100)
"Number of steps is 100"
```
Raw Strings

- The `r` prefix tells the interpreter not to transform any special characters inside the string.
- Useful, in particular, for filenames and regular expressions.

```python
>>> mydir = 'C:\test\new'
>>> print(mydir)
C:\est\new
```

```python
>>> mydir = r'C:\test\new'
>>> print(mydir)
C:\test\new
```
• Tuples are immutable lists

• Sample usages:

```python
# define a point
point = (3, -7)

# swap variables
a, b = b, a

# return more than a value
return a, b

# one-item lists
x = (10,)
```
Enumerate

- The `enumerate` built-in function enables us to iterate and count at the same time (the latter is not possible with `for` by itself)

```python
>>> a = [110, 'Ali', 'test']
>>> for i, value in enumerate(a):
...     print(i, value)
0 110
1 Ali
2 test
```
Dictionaries
Dictionaries

- Python's sole mapping type
  - keys: any immutable type
  - values: any type
- Dictionaries (a.k.a. hashes) are unordered, mutable, resizable, and iterable

```python
student = {
    'name': 'Ali',
    'id': 110
}
```
Methods

• Useful sting methods
  – keys    Keys
  – values  Values
  – items   Key-value pairs
  – get     Get value given key else default (also see in)
  – pop     Remove key from dict and return value
  – update  Update dict with contents of (an)other dict
  – copy    Make a shallow copy of dict
  – deepcopy Make a deep copy of dict
Functions
Declaring and Calling

- Functions are defined using `def` keyword

```python
def greet(name, family):
    print('Hi', name, family)

greet('Ali', 'Ajami')
greet(family='Alavi', name='Zahra')
```
Default Arguments

• Parameters can have default values

```python
def greet(name, family=''):    
    print('Hi', name, family)

greet('Ali', 'Ajami')
greet('Ali')
```
Document Strings

- A document string (aka docstring) is a string literal that occurs as the first statement in a function, and is available via `__doc__` attribute.

```python
def f(x):
    ''' Just do something on x '''
    pass

print(f.__doc__)

help(f)
```
References

- Python Official Website
  - [http://python.org/](http://python.org/)
- Python 3 Documentation
  - [http://docs.python.org/3/](http://docs.python.org/3/)
- Python Web Development with Django
  - By Jeff Forcier, Paul Bissex, Wesley Chun