Introduction to Programming Language (ITP101)

Lists, Tuples and Dictionaries

Mulualem Teku

GCIT, Bhutan

Sep 2019

...So Far & Today...

- Computational Thinking:
  - CT techniques
  - Algorithms: flowcharts / Pseudocodes
  - Input/output determination, testing algorithms (trace tables)
- Intro to Python
  - Numbers
  - Input/Output
  - Control structures
  - Functions

Next:

- Lists
- Tuples
- Dictionaries
- Sets

Brainstorm

1. Recap: What are functions? Their advantages? Iterative vs recursive functions with example?

2. Name some of the core data types (objects) in Python.

3. Mutable vs Immutable objects?

Brainstorm

1. Recap: What are functions? Their advantages? Iterative vs recursive functions with example?

2. Name some of the core data types (objects) in Python.

3. Mutable vs Immutable objects?

Brainstorm

1. Recap: What are functions? Their advantages? Iterative vs recursive functions with example?

2. Name some of the core data types (objects) in Python.

3. Mutable vs Immutable objects?

Sequence Data Types

- A.k.a. Sequences are positionally *ordered* set of objects.
- Notion of left-to-right ordering.

Some Built-in Objects

Lists ✓

Tuples ✓

```
Strings
Dictionaries              # unordered mapping type
Sets                      # unordered collection
```

- Mutable vs Immutable sequences

Sequences: 1) Lists

- Mutable ordered-collection of arbitrary objects inside ‘[ ]’.
- Offset-based access.

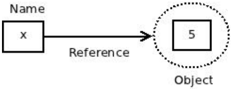
```
>>>data = [2, 4, 6, ‘hello’]
>>>print data
>>>print data[0]
>>>print data[4]                        # ??

>>>L = []                               # empty list
>>>print L

>>>matrix = [ [1,2], [3,4], [5,6] ]     # can be nested too
```

- Assignment (=) behaves differently in Python. More on this in a while.

```
>>>x = data                # Now, x too points to the list
```

Object References (Python)

- Variables link to objects. The links are a.k.a. *references*.



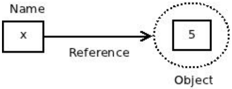
- Assignment statements do NOT copy objects.
- They manipulate object references/bindings.

Example

```
>>>x=5                                 >>>print id(x)
>>>y=x                                 >>>print id(y)
>>>x='hola'                            >>>print id(y)
>>>x, y                                >>>id(x), id(y)
```

Object References (Python)

- Variables link to objects. The links are a.k.a. *references*.



- Assignment statements do NOT copy objects.
- They manipulate object references/bindings.

Examples

```
>>>x=5
>>>y=x
>>>x=‘hola’
>>>x, y
```

```
>>>print id(x)
>>>print id(y)
>>>print id(y)
>>>id(x), id(y)
```

Example

```
>>>DNA = [‘Adenine’, ‘Thymine’]

>>>RNA = DNA

>>>RNA = ["Guanine", "Cytosine"]

>>>print (DNA, RNA)

>>>RNA = DNA

>>>RNA[1] = ‘Uracil’

>>>RNA.append(‘Protein’)

>>>print DNA, RNA
```

Example

```
>>>s = "CAGTTGGGACTAG"

>>>r = "AGTC"

>>>n = s.count(‘G’)

>>>print n

>>>r = s                      # what happens to ‘AGTC’?

>>>s = "ACGAT"

>>>print r

>>>del r

>>>print r
```

List Operations

- Concatenation (+)

```
>>>data + L
```

- Repetetion (∗)

```
>>>data * 3
```

- Indexing

- Slicing

```
>>>print data[1:-1]
>>>print data[:]
```

- Membership (`in` and `not in`)

```
>>>4 in data
>>>'hello' not in L
```

List Methods

Assume L and L2 are lists.

- `len(L)`
- `L.count(elem)`
- `L.append(elem)`
- `L.insert(index, elem)`

List Methods (2)

- L.extend(L2) # L + L2 = ?
- L.pop(index)
- L.remove(elem)
- L.index(elem)
- L.sort()
- L.reverse()

Example

```
>>>list = [‘Thymine’, ‘Guanine’]

>>>list.append(‘Cytocin’)

>>>list.insert(3,‘Adenine’)

>>>list.reverse()

>>>print list

>>>list.extend(list)

>>>list.insert(len(list),‘Uracil’)

>>>list.sort()

>>>list.pop(0)

>>>list.remove(‘Uracil’)

>>>list
```

Other Operations

- Aliasing lists

- Cloning lists

- List comprehension

Sequence Objects: 2) Tuples

- Ordered set of items inside () - optional parenthesis.

- Are *immutable* sequences, fixed-size.

```
>>>T=()
>>>T=(4,)                       # notice the comma
>>>T=(4,(6,8))                  # are nestable
```

- Offset-based access.

```
>>>T[0]
>>>T[-1]                       # negative-indexing
>>>T[99]
```

- Hetrogeneous collection

Tuple Operations

- Concatenation (+)

```
>>>T=(2,4) + (3,4)
```

- Repetetion (∗)

```
>>>T * 3
```

- Indexing

- Slicing

```
>>>T[:-1]
>>>T[1:2]
```

- Assignment

```
>>>(a,b,c)=(1,2,3)

          # size matters
```

Tuple Methods

- `len(T)`
- `T.index(elem)`
- `T.count(elem)`

- `sorted(T)    # a list`

Sorting without `sorted()`?

Tuple Methods

- `len(T)`
- `T.index(elem)`
- `T.count(elem)`

- `sorted(T)    # a list`

Sorting without `sorted()`?

```
>>>tmp=list(T)     # to mutable type
>>>tmp.sort()
>>>T=tuple(tmp)
```

Unordered Containers: a) Dictionaries

- *Mapping objects* that map/bind keys to values.

  ```
  dict = {key1:value1, key2:value2, key3:value3...}
  ```

- i.e. set of *unordered* key-value pairs.
- a.k.a. *hashes/associative arrays*.
- Key-based access, and not positional.
- Hetrogeneous, nestable
- Mutable (can shrink and grow)

Table: Greetings

Key	Value
English	Hello
Dongkha	Kuzu zangpo
Spanish	Hola
French	Bonjour
Amharic	Selam

```
>>>Greetings = { "English":"Hello", "Dongkha":"Kuzu zangpo", "Spanish":"Hola", \
"French":"Bonjour", "Amharic":"Selam" }

>>>print ( Greetings["English"] )

>>>print ( Greetings["Dzonghka"] )            # case-sensitive
```

Dictionary Operations

- Construction

```
>>>D={}                # empty dictionary

>>>D={‘title’:‘Monty Python’,‘genre’:‘comedy’}

>>>D= { (1,2,3):"hi", 4:"bye" }             # Key be immutable

>>>D={ [1,2,3]:"hi", 4:"bye"}               # Error!
```

- Lookup

```
>>>D[‘title’]

>>>D[‘year’]                         # ‘KeyError’ exception
```

Dictionary Operations

- Construction

```
>>>D={}                  # empty dictionary

>>>D={‘title’:‘Monty Python’,‘genre’:‘comedy’}

>>>D= { (1,2,3):"hi", 4:"bye" }             # Key be immutable

>>>D={ [1,2,3]:"hi", 4:"bye"}               # Error!
```

- Lookup

```
>>>D[‘title’]

>>>D[‘year’]                            # ‘KeyError’ exception
```

- Add Elements

```
>>>D[‘name’] = ‘YHWH’

>>>D[‘age’] = ‘The Ageless One’

>>>print (D)
```

- Test Membership

```
>>>‘age’ in D                          >>>‘name’ not in D
```

- Change Entry

```
>>>dict={‘country’:‘Bhutan’, ‘Dzongkhak’:‘Trashigang’}

>>>dict[‘state’]=‘Mongar’

>>>dict[‘town’]=‘Gyalpozhing’                       # ??
```

- Add Elements

```
>>>D[‘name’] = ‘YHWH’

>>>D[‘age’] = ‘The Ageless One’

>>>print (D)
```

- Test Membership

```
>>>‘age’ in D                              >>>‘name’ not in D
```

- Change Entry

```
>>>dict={‘country’:‘Bhutan’, ‘Dzongkhak’:‘Trashigang’}

>>>dict[‘state’]=‘Mongar’

>>>dict[‘town’]=‘Gyalpozhing’                          # ??
```

- Add Elements

```
>>>D[‘name’] = ‘YHWH’

>>>D[‘age’] = ‘The Ageless One’

>>>print (D)
```

- Test Membership

```
>>>‘age’ in D                           >>>‘name’ not in D
```

- Change Entry

```
>>>dict={‘country’:‘Bhutan’, ‘Dzongkhak’:‘Trashigang’}

>>>dict[‘state’]=‘Mongar’

>>>dict[‘town’]=‘Gyalpozhing’                       # ??
```

Dictionary Methods

Suppose D and D2 are dictionaries.

- `len(D)`
- `D.keys()`
- `D.values()`
- `D.items()`
- `D.copy()`
- `D.update(D2)`

- `dict.pop(key)`
- `del D[key]`
- `D.clear()`
- `del(D)`

- `dict.pop(key)`
- `del D[key]`

- `D.clear()`
- `del(D)`

Conversion from dictionaries to lists and vice versa?

Conversions: Lists & Dictionaries

- Lists from Dictionaries
  - `values()`
  - `items()`
  - `keys()`

- Dictionaries from Lists
  - `zip()`
  - `dict()`

Unordered Containers: b) Sets

Python Sets

- *Unordered* collection of *unique* and *immutable* objects.
- Neither mappings nor sequences.
- A Python implementation of the mathematical set theory.
- Members be hashable types, just like dictionary keys.
- Mutable and iterable.
- Immutable sets are called *frozensets*.
- Hetrogeneous collection.

Set Operations: Construction

1. Using the set() built-in (or frozenset() for immutables)
2. Using curly braces - {} (short-hand notation)

Examples

```
>>>S1=set("Python rocks")

>>>S1

>>>type(S1)

>>>fs=frozenset([4, 6, 8])                              # Frozenset

>>>S2={"BCA", "BScIT", "BScCS"}

>>>len(S2)                                              # cardinality

>>>S3={ (1,2,3,4,1,3,1) }                               # S3=?

>>>s4=set( ([1,2], [4,6]) )                             # Error!
```

Common Set Methods

- `S.add(elem)` `# adds immutable element`
- `S1.update(seq)`
- `S.copy()`

- `S.discard(elem)` `# if elem is not present ??`
- `S.remove(elem)` `# if elem is not present ??`
- `S.pop()` `# removes arbitrary element`
- `S.clear()` `# all elements`

Common Set Methods (2)

- `S1.intersection(S2)` ≡ `S1 & S2`
- `S1.union(S2)` ≡ `S1 | S2`
- `S1.difference(S2)` ≡ `S1 - S2`
- `S1.symmetric_difference(S2)` ≡ `S1 ∧ S2`

- `S1.issubset(S2)`
- `S1.issuperset(S2)`
- `S1.isdisjoint(S2)`