Session 8

Sequences, especially Lists, Tuples and Strings

As you arrive:

1. Start up your computer and plug it in.
2. ***Log into Angel*** and go to CSSE 120.
   Do the ***Attendance Widget*** – the PIN is on the board.
3. Go to the ***Course Schedule*** web page.
   Open the ***Slides*** for today if you wish.
4. Checkout today's project: `Session08_Sequences`

Sequences

- What are they, why use them?
- Types of sequences: Lists, Tuples, Strings, and more

Sequences

- Looping through a sequence
- Accumulating a sequence

Checkout today's project:

`Session08_Sequences`

Are you in the Pydev perspective? If not:

`Window ~ Open Perspective ~ Other` then `Pydev`

Messed up views? If so:

`Window ~ Reset Perspective`

Troubles getting today's project? If so:

No SVN repositories view (tab)? If it is not there:

`Window ~ Show View ~ Other`
then `SVN ~ SVN Repositories`

1. ***In your SVN repositories view (tab), expand your repository*** *(the top-level item) if not already expanded.*
   - If no repository, perhaps you are in the wrong Workspace. Get help.
2. ***Right-click on today's project****, then select* `Checkout`*.* *Press **OK** as needed.* The project shows up in the
   `Pydev Package Explorer`
   to the right. Expand and browse the modules under `src` as desired.

Outline of today's session

Checkout today's project: `Session08_Sequences`

- ***Sequences***
  - ***What is a sequence?***
    - Why is it so powerful?
    - How to reference its items with the ***square-bracket*** notation
  - ***Kinds of sequences***
    - Six kinds in Python: ***lists***, ***tuples***, ***strings***, ***bytes***, ***byte arrays***, ***ranges***
  - ***Loop through a sequence***
    - Directly
    - With indices generated by a range expression
      - Variation: the loop references other indices too
  - ***Accumulate a sequence***
    - With the + operator
    - With ***append*** (for lists) and ***join*** (for strings)
  - ***Mutating sequences***
  - ***Methods and functions for sequences***

Next time

Data types

- *Data*
  - Information stored and manipulated on a computer
  - Ultimately stored as bits – 0s and 1s
- But the type of each data item determines:
  - How to interpret the bits
- *Data type*
  - A particular way of interpreting bits
  - Determines the possible values an item can have
  - Determines the operations supported on items
  - Python types include: *int*, *float*, *str*, *list*, *function*, *tuple*

1. ***Sequence*** – what is it (in Python)?

- A ***sequence*** is a type of thing in Python that represents an entire *collection* of things.
- More carefully, it represents a
  - finite • *ordered* • *collection* of things
  - indexed by whole numbers

There are also types for **UNordered collections** *of things –* **sets** *and* **Circles,** *for example. More on these in a subsequent session.*

- Examples:
  - A ***list*** `["red", "white", "blue"]`
  - A ***tuple*** `(800, 400)`
  - A ***str*** **(string)** `"Check out Joan Osborne, super musician"`

2. Why are Sequences powerful?

- **A sequence lets you refer to an entire collection using a *single name*.**
- You can still get to the items in the collection, by ***indexing***:

```
    colors = ["red", "white", "blue"]
    colors[0]       has value "red"
    colors[1]       has value "white"
    colors[2]       has value "blue"
```

Indexing starts at ZERO, not at one.

- And you can *loop* through the items in the collection, like this:

```
    for color in colors:
        circle = zg.Circle(...)
        circle.setFill(color)
```

3. Types of Sequences

- There are currently 6 built-in types of Sequences, in two flavors:

Mutable:
- `list`
- `bytearray`

Immutable:
- `str` (a *string*)
- `tuple`
- `range`
- `bytes`

Mutable: *the collection can change after it is created:*
- *Its items can change.*
- *Items can be deleted and added.*

Immutable: *once the collection is created, it can no longer change.*

The following slides explain that different types of Sequences differ in their:
- ***mutability***
- ***type of things they can contain***
- ***notations** / how you make them*
- ***operations** that you can do to them*

*These are just the **built-in** Sequence types, that is, the ones that you can use without an `import` statement. The `array` and `collections` modules offer additional mutable Sequence types.*

4a. Mutability

This and the following slides explain that different types of Sequences differ in their:

- ***mutability***
- ***type of things they can contain***
- ***notations** / how you make instances*
- ***operations** that you can do to them*

- Lists are mutable:

```
colors = ["red", "white", "blue"]
colors[1] = "grey"
colors.append("bob")
```

OK

`colors` becomes
`["red", "grey", "blue"]` then
`["red", "grey", "blue". "bob"]`

- Strings and tuples are NOT mutable:

```
building = "Taj Mahal"
building[2] = "g"
pair = (48, 32)
pair[0] = 22
```

NOT OK.
Gives an error message when executed.

- The following (which continue the example from the previous bullet) have nothing to do with mutability and are perfectly OK:

```
building = "Sistene Chapel"     pair = (0, 0)     colors = []
building = building.replace("Mahal", "Begum")
```

4b. Things that Sequences can contain

This and the following slides explain that different types of Sequences differ in their:

- ***mutability***
- ***type of things they can contain***
- ***notations** / how you make instances*
- ***operations** that you can do to them*

Type	What objects of this type can contain
list	anything
bytearray	bytes, that is, *integers* between 0 and 255
***str* (a string)**	Unicode characters (each 16 or 32 bits, depending on an installation option)
tuple	anything
range	ranges generated by `range`
bytes	Bytes (*integers* between 0 and 255)

A ***bit*** is a 0 or 1.

Each ***byte*** is 8 bits and represents an ASCII encoding of one of the 128 pre-Unicode characters.

Unicode allows for far more than the 128 ASCII characters and is the modern standard. See pp. 132-133 or your text.

If you ever need a list-like thing that holds only (say) int's, check out the `array` *module.*

4c. Notation and how you can make *instances*

This and the following slides explain that different types of Sequences differ in their:

- ***mutability***
- ***type of things they can contain***
- ***notations / how you make instances***
- ***operations** that you can do to them*

Type	**Notation, and how you make an instance** (options, but not ALL of the options, are shown here)

list	`[`*blah, blah, ...*`]` `list(`*`sequence`*`)`
	`[`*`expression`*` for `*`variable`*` in `*`sequence`*`]`

str **(a string)**	`"the charac'ters"` `'the charac"ters'`
	`'''characte\\rs in a \a string with \xF9`
	`stuff th\o274at br\'eaks across lines.'''`

tuple	`(`*blah, blah, ...*`)` *blah, blah, ...* But special cases for 0 or 1 elements: `()` `(`*blah*`,)`
range	`range(`*`m`*`)` `range(`*`m`*`, `*`n`*`)` `range(`*`m`*`, `*`n`*`, `*`i`*`)`

4c. Notation and how you can make *instances* (continued)

This and the following slides explain that different types of Sequences differ in their:

- ***mutability***
- ***type of things they can contain***
- ***notations / how you make instances***
- ***operations** that you can do to them*

Type	Notation, and how you make an instance (options, but not ALL of the options, are shown here)
bytes	*Same as for strings, but put a* **b** *in front, e.g.* `b"the charac'ters"` `b'the charac"ters'` `bytes(`*list of ASCII codes*`)` *For example,* `b'rat'` *is the same as* `bytes([114, 97, 116])`
bytearray	`bytearray(`*bytes object*`)` `bytes(`*list of ASCII codes*`)`

4d. Operations that you can do to Sequences

- We'll discuss these in the NEXT session

This and the following slides explain that different types of Sequences differ in their:

- ***mutability***
- ***type of things they can contain***
- ***notations** / how you make instances*
- ***operations that you can do to them***

Exercises

- Do m1, m2, and m3. Then do m5.
  - Save some of m5 for homework, perhaps.
- Do m4. Then do m6.
  - Finish m6 for homework