Beyond the Textbook (Zelle 3e - Chapter 5)

Sequence-Like Python Types

Strings, Lists, Tuples, Ranges, and *Files*

Commonly Used Python Sequence Types

Type	Allowed Item Types	Mutable
str	*character*	✗
list	Any	✓
tuple	Any (immutable type preferred)	✗
range	int	✗

Python `file` is Sequence-Like

- The **file type** is not a Python sequence in the narrowest sense.
- Nevertheless, because a file is an ***iterable***, Python file objects implement **For-In**.
- Files are included in the Zelle *Sequences* chapter to emphasize their synergy with formal sequence types like list and tuple.

Features of Commonly Used Python Sequence Types

Type	Mutable	Iterate	Unpack	Index	Slice	Extended Slice	del	methods
range	✗	✓	✓	✓	✓	✓	✗	✓
str	✗	✓	✓	✓	✓	✓	✗	✓
list	✓	✓	✓	✓	✓	✓	✓	✓
tuple	✗	✓	✓	✓	✓	✓	✗	✓

Iteration

- Strings, lists, tuples, ranges, and files are *iterables*.
- All of these support the Python **For-In** construct.
- See `_10_demonstrate_iteration.py` in the supplemental project.
- Even lists (which are mutable) cannot be updated using the For-In construct.
- Instead, lists are updated using index expressions.
- See `_12_demonstrate_updating_lists.py` in the supplemental project.

Unpacking

- Python iterables support **unpacking**.
- Unpacking is implemented with an assignment statement.
- **Syntax:** *scalar_1, scalar_2 ... = iterable*
- Unpacking is a good coding practice that improves code readability.
- See `_15_demonstrate_unpacking.py` in the supplemental project.

Indexing

- Python strings, lists, tuples, and ranges support **indexing**.
- Indexing provides access to individual items in the sequence.
- Individual items can be referenced using index expressions.
- **Syntax:** *variable_name*[*index_number*]
- The **first** item in the sequence has index **0**.
- The **nth** item in a sequence has index **n - 1**.
- These positive index values are numbered from **left to right**.
- There are also negative index values that are numbered from **right to left**.

String Indexing Illustrated

	G	r	a	c	e		H	o	p	p	e	r	
	0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

```
>>> person = 'Grace Hopper'
>>> person
'Grace Hopper'
>>> person[0]
'G'
>>> person[11]
'r'
>>> print(person[0], person[6])
G H
>>> index_value = 6
>>> person[index_value + 2]
'p'
>>>
```

```
>>> person
'Grace Hopper'
>>> person[-12]
'G'
>>> person[-1]
'r'
>>> print(person[-12], person[-6])
G H
>>> index_value = -6
>>> person[index_value + 2]
'p'
>>>
```

```
>>> person
'Grace Hopper'
>>> person[0]
'G'
>>> person[0] = 'X'
Traceback (most recent call
last):
 File "<stdin>", line 1, in
<module>
TypeError: 'str' object does
not support item assignment
>>>
```

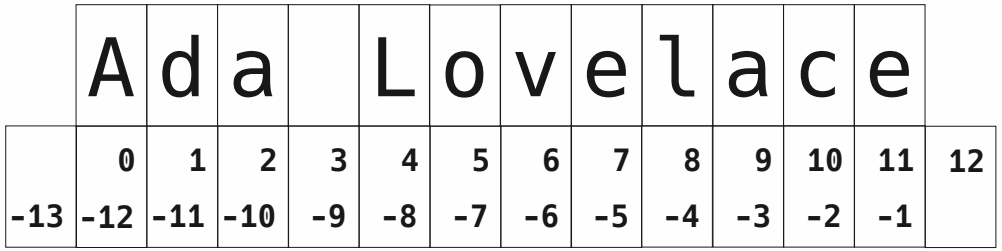
More Indexing Examples

- Indexing also works with lists, tuples, and ranges.
- See `_20_demonstrate_indexing.py` in the supplemental project.

Slicing

- Python strings, lists, tuples, and ranges support slicing.
- Slicing allows for the extraction of **sub-sequences**.
- **Syntax:** *variable_name[start : stop]*
- *start* is the index of the first item to be included.
- *stop* is **one higher** than the index of the last item to be included.
- Omitting *start* causes sub-sequence to start at beginning of sequence.
- Omitting *stop* causes sub-sequence to continue until end of sequence.
- Note that the positive and negative index values create synonyms.
- You can interchange positive and negative index values when convenient.

String Slicing Illustrated




```
>>> person = 'Ada Lovelace'
>>> person
'Ada Lovelace'
>>> person[0 : 12]
'Ada Lovelace'
>>> person[ : ]
'Ada Lovelace'
>>> person[ : 3]
'Ada'
>>> person[4 : ]
'Lovelace'
>>>
```

```
>>> person
'Ada Lovelace'
>>> person[0 : -1]
'Ada Lovelac'
>>> person[0 : -4]
'Ada Love'
>>> person[4 : 8]
'Love'
>>> person[4 : -4]
'Love'
>>> person[-8 : -4]
'Love'
>>>
```

```
>>> person
'Ada Lovelace'
>>> person[-12 : -9]
'Ada'
>>> person[-12 : -9] = 'Xxx'
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
TypeError: 'str' object does not sup-
port item assignment
>>>
```

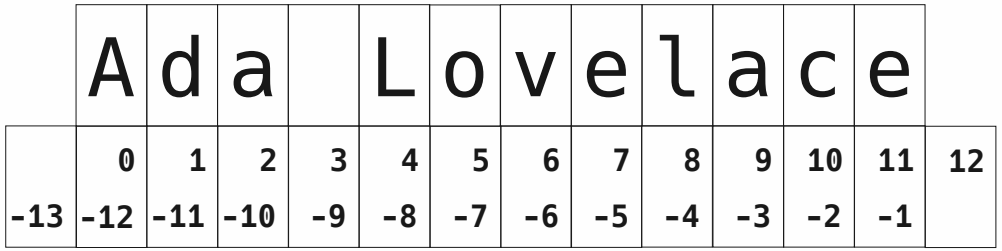
More Slicing Examples

- Slicing also works with lists, tuples, and ranges.
- See `_25_demonstrate_slicing.py` in the supplemental project.

Extended Slicing

- Python strings, lists, tuples, and ranges support **extended slicing**.
- Extended slicing provides for an additional step argument.
- **Syntax:** *variable_name[start : stop : step]*
- *start* and *stop* work the same as in regular slicing.
- Postivite *step* values cause extraction **from left to right**.
- A *step* of 1 extracts characters consecutively.
- A *step* of 2 extracts every other character from the string.
- A *step* of 3 extracts every third character from the string.
- Negative *step* values cause extraction **from right to left**.

Extended String Slicing Illustrated




```
>>> person = 'Ada Lovelace'
>>> person[0 : 12 : 1]
'Ada Lovelace'
>>> person[ : : 1]
'Ada Lovelace'
>>> person[ : : 2]
'AaLvlc'
>>> person[ : : 3]
'A va'
>>>
```

```
>>> person
'Ada Lovelace'
>>> person[ : : -1]
'ecalevoL adA'
>>> person[ : : -2]
'eaeo d'
>>> person[ : : -3]
'eloa'
>>> person[ : : -3] = 'xxx'
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support
item assignment
```

```
>>> person
'Ada Lovelace'
>>> person[0 : 12 : -1]
''
>>> person[11 : -13 : -1]
'ecalevoL adA'
>>> person[11 : -13 : -2]
'eaeo d'
>>> person[11 : -13 : -3]
'eloa'
>>>
```

More Extended Slicing Examples

- Extended slicing also works with lists, tuples, and ranges.
- See `_30_demonstrate_extended_slicing.py` in the supplemental project.

The `del` Operator

- Python lists are **mutable**.
- So items within lists may be deleted.
- One way to delete an item in a list is to use the **del** operator.
- `del` statements may use index expressions.
- **Syntax:** del *variable_name*[*index_number*]
- `del` statements may use slice expressions.
- **Syntax:** del *variable_name*[*start*, *stop*]
- See `_35_demonstrate_delete.py` in the supplemental project.

Methods

- Python strings, lists, tuples, and ranges have methods
- See Python documentation for a complete list of methods available.

Some `str` Class Methods

Method	Description
str.upper()	Returns new string shifted to upper case.
str.lower()	Returns new string shifted to lower case.
str.title()	Returns new string shifted to title case.
str.strip()	Returns a new string with leading and trailing characters removed.
str.split()	Returns a list of substrings created by splitting.
str.find()	Returns the lowest index of a substring.
str.count()	Returns a count of the occurrences of a substring.

Some `list` Class Methods

Method	Description
list.append()	Add an item to the end of the list.
list.insert()	Insert an item at a given position.
list.pop()	Remove an item at a given position, and return it.
list.remove()	Remove an item with a particular value from the list.
list.clear()	Remove all items from the list.

More `list` Class Methods

Method	Description
list.index()	Returns the index of the first item that matches a particular value.
list.count()	Returns the number of times a particular value occurs in the list.
list.sort()	Sorts the items in the list in place.
list.reverse()	Reverses the order of the items in the list in place.
list.copy()	Returns a shallow copy of the list.

Tuple Basics

- Python tuples are **sequences** of items (value objects).
- Tuples may contain either **homogenous** or **heterogenous** items.
- Tuples have a variable length (including empty).
- Tuples are **immutable**.
- The items in a tuple may themselves be either **mutable** or **immutable**.
- Using **immutable types** for tuple items is considered a best practice.
- Tuples are often used to hold related data facts that make up a **record**.
- Python functions that appear to return more than one value actually return a tuple.

Some `tuple` Class Methods

Method	Description
tuple.count()	Works like count method of list class
tuple.index()	Works like index method of list class

More Python String Features

- Strings support **escape sequences** that allow entry of values not easily entered at the keyboard. See https://python-reference.readthedocs.io/en/latest/docs/str/escapes.html
- f-strings and other string formatting approaches support the Format Specification Mini-Language. See https://docs.python.org/3/library/string.html?highlight=str#format-specification-mini-language

More Python File Features

- When calling the `open` function, it is a best practice to include the `encoding='utf-8'` **keyword parameter**.
- Opening a file with a mode of `a` causes records written to be appended to the end of an existing file.

Extra Python Features (Syntactic Sugar)

See https://en.wikipedia.org/wiki/Syntactic_sugar

`with` Statement

- The `with` statement is a tool that creates a **context manager**.
- Context managers help manage system resources like files.
- Rather than having to close each open file explicitly, a program can use the `with` statement to close them implicitly.
- Some intermediate and advanced Python programmers prefer this approach to managing file closing.
- See `_40_demonstrate_with_using_single_file.py` in the supplemental project.
- See `_45_demonstrate_with_using_multiple_files.py` in the supplemental project.

String Method Chaining

- Some Python classes allow their methods to be called in series using a special syntax called **method chaining**.
- Method chaining allows the output of one method call to be used as input to the next method call.
- It sets up a **pipeline** of method calls.
- **Syntax:** `result = variable.method_1().method_2().method_3()`
- Many data science packages (including **pandas**) make extensive use of method chaining syntax.
- See `_50_demonstrate_string_method_chaining.py` in the supplemental project.

Resources

Martelli, A., Ravenscroft, A. M., & Holden, S. (2017). *Python in a nutshell* (Third edition). O'Reilly Media.

Zelle, J. M. (2017). *Python programming: An introduction to computer science* (Third edition). Franklin, Beedle & Associates Inc.

Last Revised 2022-09-05