

# SI 506 Lecture 06

---

## Topics

1. Slicing, part II
2. Select `str` and `list` methods
3. String formatting
4. In-class coding challenges

## Vocabulary

- **Concatenation.** Joining one object to another in order to create a new object. Joining two strings together (e.g., `greeting = 'Hello ' + 'SI 506'`) is an example of string concatenation.
- **Index.** Numeric position of an element or item contained in an ordered sequence. Python indexes are zero-based, i.e., the first element's index value is 0 not 1.
- **Iterable.** An object capable of returning its members one at a time. Both strings and lists are examples of an iterable.
- **Sequence.** An ordered set such as `str`, `list`, or `tuple`, the members of which (e.g., characters, elements, items) can be accessed individually or in groups.
- **Slice.** A subset of a sequence. A slice is created using the slice notation operator `[]` with colons separating numbers when several are given, such as in `variable_name[1:3:5]`. The bracket notation uses slice objects internally.
- **Slice notation operator.** Square brackets `[]` enclosing either an index value or a slicing expression that is used to access individual or groups of sequence characters, elements or items.
- **Tuple.** An ordered sequence that cannot be modified once it is created.

## Reference

Open the following [w3schools](#) reference pages in your browser and bookmark them. The pages provide useful summaries of `str`, `list`, and `tuple` methods.

1. w3schools, ["Python String Methods"](#)
2. w3schools, ["Python List Methods"](#)
3. w3schools, ["Python Tuple Methods"](#).

## Lecture data

Today's lecture features data derived from various Monty Python comedy sketches including the Pythons' famous ["Spam" sketch](#) (1970) including the Spam dominated cafe [menu](#).

During the Second World War and after Britain imposed rationing restrictions and, starting in 1941, imported massive quantities of canned spam from the United States as a protein substitute for imports of beef, pork,

and poultry. The public, including my parents, grew to loathe it--which the sketch plays upon in surrealist fashion.

💡 Have you ever wondered why unwanted email is referred to as "spam". Watch the ["Spam" sketch](#) and you'll quickly understand why.

## 1.0 Slicing, part II

Recall from the previous lecture that you can access a subset or *slice* of a sequence using Python's slice notation.

A slicing operation starts at index position zero (`0`) and returns all members of a sequence up to but excluding the member located at the specified `end` position. An optional `start` position can be specified in order to start the slice at an index other than zero (`0`). Negative values can be employed to commence the slice from the end as opposed to the beginning of the sequence.

An optional `stride` number can also be specified in order to skip members or reverse the slice. A positive `stride` value will return every *n*th member in the slice range; inclusion of a negative `stride` value will return every *n*th member in the slice range in *reverse* order. The default `stride` value equals `1`.

```
some_sequence[< optional_start >: < required_end >: < optional_stride >]
```

## 1.1 Working with stride values

An optional `stride` value can be specified to *increase* the number of slicing steps in order to *skip* one or more elements, items, or characters in a sequence. Providing a negative stride value *reverses* the operation.

Utilize the Spam sketch `cast` list to practice slicing with `stride` values.

```
cast = [  
    'Terry Jones, Waitress',  
    'Eric Idle, Mr Bun',  
    'Graham Chapman, Mrs Bun',  
    'John Cleese, The Hungarian',  
    'Michael Palin, Historian',  
    'Extra, Viking 01',  
    'Extra, Viking 02',  
    'Extra, Viking 03',  
    'Extra, Viking 04',  
    'Extra, Viking 05',  
    'Extra, Viking 06',  
    'Extra, Police Constable'  
]
```

## 1.1.1 Challenge 01 (warm up)

**Task:** Retrieve named cast members.

1. Employ slicing *with a `stride` value* to retrieve the named cast members in the `cast` list.
2. Assign the new list to the variable `cast_members`. Uncomment `print()` and check your work.

## 1.1.2 Challenge 02

**Task:** Retrieve deceased cast members.

1. Employing slicing with a `stride` value to retrieve the two deceased members of the Spam sketch cast:
  - Terry Jones (1942-2020)
  - Graham Chapman (1941-1989)
2. Assign the new list to the variable `cast_members`. Uncomment `print()` and check your work.

## 1.1.3 Challenge 03

**Task:** Return the cast members in *reverse* order.

1. Employing slicing with a `stride` value to return the cast members of the `cast` list in reverse order.
2. Assign the new list to the variable `cast_members`. Uncomment `print()` and check your work.

## 1.1.4 Challenge 04

**Task:** Return every other cast member.

1. Employing slicing with a `stride` value to return *every other* cast member of the `cast` list.
2. Assign the new list to the variable `cast_members`. Uncomment `print()` and check your work.

## 1.1.5 Challenge 05

**Task:** Return every other cast member in reverse order.

1. Employing slicing with a `stride` value to return *every other* cast member of the `cast` list in reverse order.
2. Assign the new list to the variable `cast_members`. Uncomment `print()` and check your work.

## 1.1.6 Challenge 06

**Task:** Return every other Viking.

1. Employing slicing with a `stride` value to return *every other* Viking cast member of the `cast` list.
2. Assign the new list to the variable `cast_members`. Uncomment `print()` and check your work.

## 1.2 Slice Assignment

Note that can assign a new value to a list element by referencing its index position.

```
cast[4] = 'Michael Palin, The Historian' # Adds the definite article "The"
```

You can also replace a subset of a list with another list or subset of a list using slice assignment.

```
mounties = [  
    'Extra, Canadian Mountie 01',  
    'Extra, Canadian Mountie 02',  
    'Extra, Canadian Mountie 03',  
    'Extra, Canadian Mountie 04',  
    'Extra, Canadian Mountie 05',  
    'Extra, Canadian Mountie 06'  
]
```

```
# Replace part of a list (length unchanged).  
cast[5:11] = mounties[0:] # replace Vikings with Mounties
```

```
# Replace part of a list (length changes).  
cast[5:11] = mounties[1:5] # replace Vikings with Mounties 02-04
```

## 1.3 Built-in del() function and slicing

You can employ slicing and the built-in `del()` function to remove subsets of a sequence.

```
# Delete the Mounties (retain the Police Constable)  
del(cast[-5:-1])  
# del(cast[5:9]) # alternative
```

## 1.4 built-in slice() function

You can also use the built-in `slice()` function to return a slice object and apply it to a sequence. `slice()` accepts three arguments: an optional integer `start` value (default = 0), a required integer `end` value that specifies the position in which to end the slicing operation, and an optional `step` value that specifies the slicing step (default = 1).

```
# slice([start, ]end[, step]) object
s = slice(1, 4, 2)
cast_members = cast[s] # Returns Idle and Cleese
```

## 2.0 String and list methods

When you create a string or list you create an object that is based on a `class`, a type that can both hold data and perform actions. Think of a `class` as a template, blueprint, or model for creating objects. Each string or list that you create and assign to a variable represents an *instance* of the class upon which it is based (i.e., the class types `str` and `list`). Such objects possess individual characteristics (data) and common behaviors (methods).

Object methods, if defined, are *called* using dot (`.`) notation. If a method "signature" includes one or more *parameters*, these can be passed to the method as comma-separated *arguments* that are included inside the method name's parentheses. If a method definition does not specify an *explicit* return value, `None` (type: `NoneType`) will be returned implicitly to the caller.

```
menu_item = 'Spam, egg, Spam, Spam, bacon and Spam'

# str.lower() -- no argument method
menu_item_lower = menu_item.lower()

# str.count(value, start=0, end=len(str) - 1) -- start and end are optional
spam_count = menu_item.count('Spam')

# str.split(sep=' ', maxsplit=-1) -- sep and maxsplit are optional
items = menu_item.split(', ') # returns list

# list.remove(element) -- in-place operation; removes 1st occurrence; returns None
items.remove('egg')

# Do not do this: items variable no longer points to a list object
items = items.remove('bacon and Spam') # None is returned
```

## 2.1 Chained method calls

Method calls can also be "chained". Each method call returns a value (object) to which the next method call is bound. Order matters. Note that calling a method not defined for an object will raise an `AttributeError`.

```
menu_item = 'Egg, bacon, sausage and Spam'

# Good. Replace, convert to lower case, and split.
items = menu_item.replace(' and', ',').lower().split(', ')

# Bad. The trailing list.append() returns None (oops!)
items = menu_item.replace(' and', ',').lower().split(', ')
items.append('pancakes')

# Ugly. Premature split. Calling lower on a list object raises a runtime
error
# AttributeError: 'list' object has no attribute 'lower'
items = menu_item.replace(' and', ',').split(', ').lower()
```

## 2.2 Select `str` methods

String objects (type `str`) include "built-in" behaviors that are defined as *methods*. Calling a string method may require that one or more arguments (values) be passed to it in order to perform the requested computation. Depending on the method definition, the operation may result in a computed value being returned to the caller; otherwise `None` is returned (implicitly).

Earlier lectures introduced the following string methods:

- `str.count()`
- `str.lower()` / `str.upper()`
- `str.startswith()` / `str.endswith()`
- `str.replace()`
- `str.split()` / `str.splitlines()`

Below is another group of string methods that you should get to know.

### 2.2.1 `str.strip()`

Returns a "trimmed" version of the string, removing leading and trailing spaces as well as newline escape sequences (`\n`).

```
monty_python = " Monty Python's Flying Circus \n"  
monty_python = monty_python.strip()
```



To remove spaces from either the beginning or the end of the string only use `str.lstrip()` or `str.rstrip()`.

## 2.2.2 str.find()

Finds the *first* occurrence of the specified value and returns its index value. If the value is not located -1 is returned.

```
menu_item = 'Spam, Spam, Spam, egg and Spam'  
position = menu_item.find('Spam')  
  
menu_item = 'Spam, Spam, Spam, egg and Spam'  
position = menu_item.find('ham')
```



`str.rfind()` attempts to locate the *last* occurrence of the specified value. If the value is not located -1 is returned.

## 2.2.3 str.index()

Finds the *first* occurrence of the specified value and returns its index value. If the value is not located a `IndexError` is raised.

```
menu_item = 'Spam, Spam, Spam, egg and Spam'  
position = menu_item.index('egg and Spam')  
  
# TODO UNCOMMENT and raise runtime exception  
# position = menu_item.index('ham') # IndexError
```

## 2.2.4 str.join()


Returns a new string by joining each element in the passed in *iterable* to the specified string.

```
items = ['Oatmeal', 'Fruit', 'Spam'] # a list  
menu_item = ' '.join(items) # build a string by joining each element to an  
empty string
```

```
menu_item = ', '.join(items) # build a string by joining each element to a
comma
```

## 2.2.5 Challenge 07

**Task:** The Python's cafe in Bromley is under new management. Revise the cafe `menu` employing various string methods.

 Break the problem into subproblems solving each subproblem in turn by calling a `str` method. Uncomment the `print()` function and run your file after every change, printing `menu_v2` to the terminal screen in order to check your work.

Implement the following changes to the multiline string named `menu`. Use **method chaining** to accomplish the task.

### Requirements

**!** The requirements are *unordered* and may not represent the correct order of operations to perform.

- Replace every instance of the substring `sausage` with the string `toast`.
- Substitute every **third (3rd) helping** of `Spam` listed in a menu item with the value assigned to the variable `healthy_choice`.

Example substring substitution:

`'Spam, Spam, Spam, egg and Spam' -> 'Spam, Spam, Oatmeal, egg and Spam'`

- Split the multiline string into a list and assign the new menu to the variable named `menu_v2`.
- Convert the menu to lower case.

```
menu = """Egg and bacon
Egg, sausage and bacon
Egg and Spam
Egg, bacon and Spam
Egg, bacon, sausage and Spam
Spam, bacon, sausage and Spam
Spam, egg, Spam, Spam, bacon and Spam
Spam, Spam, Spam, egg and Spam
Spam, Spam, Spam, Spam, Spam, Spam, baked beans, Spam, Spam, Spam and Spam
Lobster Thermidor aux crevettes with a Mornay sauce, garnished with
truffle pâté, brandy and a fried egg on top and Spam"""

healthy_choice = 'Oatmeal'

menu_v2 = None # TODO Implement
```



## 3.0 List methods

As with strings, List objects (type: `list`) also include "built-in" behaviors that are defined as *methods*. Calling a `list` method may require that one or more arguments (values) be passed to it in order to perform the requested computation. Depending on the method definition, the operation may result in a computed value being returned to the caller; otherwise `None` is returned.

Below is a select list of `list` methods.

### 3.0.1 `list.append()`

Appends element to the end of a list. The operation mutates the list *in-place*, returning `None` implicitly to the caller.

```
menu_v2.append('red beans and rice') # modify in-place (no variable assignment)
```

! Do not assign the return value of `list.append(< element >)` to the current list variable. Doing so will assign `None` to the variable.

### 3.0.2 `list.remove()`

Remove element from list. The operation mutates the list *in-place*, returning `None` implicitly to the caller.

```
item = menu_v2[-2] # Lobster Thermidor  
menu_v2.remove(item)
```

### 3.0.3 `list.extend()`

Extend list with another list. The operation mutates the list *in-place*, returning `None` implicitly to the caller.

```
healthy_items = ['cereal, yogurt, and spam', 'oatmeal, fruit plate, and  
spam']  
menu_v2.extend(healthy_items)
```

### 3.0.4 `list.index()`

Return index position by value.

```
index = menu_v2.index('egg, bacon, and spam')
```

### 3.0.5 list.insert()

Insert element at specified index position. The operation mutates the list *in-place*, returning **None** implicitly to the caller.

```
menu_v2.insert(1, 'belgian waffle, strawberries, and spam')
```

### 3.0.6 list.sort()

Sort the list. The operation mutates the list *in-place*, returning **None** implicitly to the caller.



You can pass the optional arguments **reverse=True|False** (pipe = 'or') as well as **key=some\_function** in order to further specify the sorting criteria (out of scope for the moment).

```
menu_v2.sort() # default alpha sort
```

## 3.1 Challenge 08

**Task:** Create yet another version of the Bromley cafe menu.

1. Use slicing to create a new list by reversing the order of **menu\_v2**. Assign the new list to a variable named **menu\_v3**. Uncomment **print()** and check your work.



the list method **list.reverse()** reverses the order of a list's elements "in-place"; in other words it mutates an *existing* list rather than returning a new list to the caller.

2. Use the appropriate list method to eliminate the **menu\_v3** item 'egg and spam'. Uncomment **print()** and check your work.
3. Use the appropriate list method to add 'blueberry pancakes' to **menu\_v3** in the sixth (6th) position. Uncomment **print()** and check your work.

## 3.2 Challenge 09

**Task:** Return a slice of the new Bromley cafe menu.

1. Use the appropriate list method to return the index value of the menu item **"egg, toast and bacon"**. Assign the return value to a variable named `idx`. Uncomment `print()` and check your work.
2. Use the appropriate built-in function to return the length of `menu_v3`. Assign the return value to a variable named `length`. Uncomment `print()` and check your work.
3. Utilize the `idx` and the `length` values in a slicing expression that returns a subset of `menu_v3` that contains *only* those menu items that list "egg" *first*. Assign the new list to the variable named `egg_first_dishes`. Uncomment `print()` and check your work.



You can calculate a slice's `start`, `end`, and/or `stride` value by embedding an arithmetic expression inside the slice notation's subscript operator `[]`.

```
menu_v3[< start >:< end arithmetic >]
```

The new list that you must produce:

```
egg_first_dishes = [  
    'egg, toast and bacon',  
    'egg, bacon, toast and spam',  
    'egg, bacon and spam',  
    'egg and bacon'  
]
```

## 4.0 String formatting

There are three ways to format one or more values as a string. We recommend that you utilize the newest approach: the *formatted string literal* (f-string). That said, you will encounter the other string formatting routines when reading older code or tutorials so its important to understand how to implement the other approaches.

### 4.1 Formatted string literal (f-string)

The f-string syntax `f"some_string {some variable}"` is less verbose and easier to construct than earlier string formatting approaches. Employ curly braces to denote embedded variables in the expression.

```
special_item = 'egg, bacon, spam and sausage'  
question = f"Why can't she have {special_item}?" # embedded variable
```

💡 Recall that `\n` represents an escape sequence, specifically an ASCII linefeed (LF). Think of `\n` as "new line". Passing `\n` in a string will insert a new line at the position of the escape sequence.

## 4.2 str.format()

Formats the specified value(s) and inserts them inside the string's using curly braces `{}` as a placeholder.

```
question = "Could I have {}, {}, {} and {}, without the  
spam?".format('egg', 'bacon', 'spam', 'sausage')
```

💡 The placeholders can be identified using empty placeholders `{}`, numbered indexes `{0}`, or named indexes `{egg}`.

## 4.3 C-style or simple positional formatting

The oldest of the three string formatting approaches. Uses the `%` character as a placeholder.

Placeholders (select list):

- `%c` = single character placeholder
- `%d` = decimal placeholder
- `%i` = integer placeholder
- `%s` = string placeholder

```
question = "No, it wouldn't be %s, %s, %s and %s, would it?" % (egg,  
bacon, spam, sausage)
```

For a summary of ye olde C-style / simple positional formatting see Frank Hofman, ["Python String Interpolation with the Percent \(%\) Operator"](#) (Stack Abuse, nd).