Objectives

- Continuing text processing, manipulation
  - String methods, operations, processing

In the News: Air Canada's Chatbot

Air Canada Has to Honor a Refund Policy Its Chatbot Made Up

The airline tried to argue that it shouldn't be liable for anything its chatbot says.

- 2022: Chatbot gave bad information about bereavement policy, along with link to correct info
- Air Canada: "the chatbot is a separate legal entity that is responsible for its own actions."
- Judge: nope, you gotta pay; how should they know?

Review

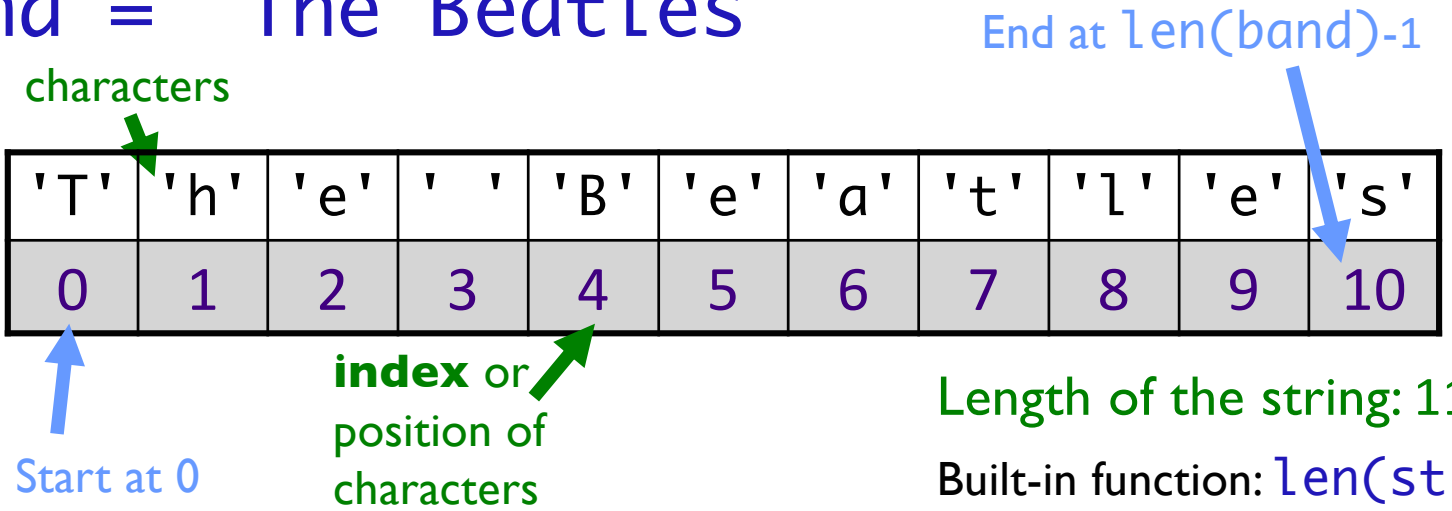
Get out your handouts from last time

- How do we represent text?
  - How can we represent really long text?
- How can we combine strings?
  - How can we combine strings multiple times?
  - What if we want to combine a string and an integer? What do we need to do?
- How can you find out how many characters are in a string?
- How do we find the character at a particular position of a string?
- Object-oriented programming:
  - What is an API?
  - What are methods?
  - How do we call methods on an object?
- What is your favorite `str` method?
- What does it mean to say that strings are ***immutable***?
  - i.e., what are the consequences of strings being immutable?

Review: Strings

- A *sequence* of one-character strings
  - Example:

```
band = "The Beatles"
```



Length of the string: 11

Built-in function: `len(string)`

to find length of a string

Substrings Operator: []

Literally, **not** optional

- Look at a particular character in the string
  - Syntax: `string[<integer_expression>]`
  - [Positive value]: index of character
  - [Negative value]: count backwards from end
- Examples:
  - `<sequence>[0]` returns the first element/char
  - `<sequence>[-1]` returns the last element/char

We will deal with sequences beyond strings later.

Examples in interpreter

Common `str` Methods

Method	Operation
`center(width)`	Returns a copy of string centered within the given number of columns
`count(sub[, start [, end]])`	Returns # of non-overlapping occurrences of substring `sub` in the string.
`endswith(sub)` `startswith(sub)`	Returns `True` iff string ends with/starts with `sub`
`find(sub[, start [, end]])`	Returns first index where substring `sub` is found
`isalpha(), isdigit(), isspace()`	Returns `True` iff string contains letters/digits/whitespace *only*
`lower(), upper()`	Returns a copy of string converted to lowercase/uppercase

Common `str` Methods

Method	Operation
`replace(old, new[, count])`	Returns a copy of string with all occurrences of substring `old` replaced by substring `new`. If `count` given, only replaces first `count` instances.
`split([sep])`	Returns a list of the words in the string, using `sep` as the delimiter string. If `sep` is not specified or is None, any whitespace string is a separator.
`strip()`	Returns a copy of the string with the leading and trailing whitespace removed
`join(<sequence>)`	Returns a string which is the concatenation of the strings in the sequence with the string this is called on as the separator
`swapcase()`	Returns a copy of the string with uppercase characters converted to lowercase and vice versa.

Functions vs Methods (with Strings)

Functions

- Associated with a file or module
- All input comes from arguments/parameters
- Example: `len` is a built-in function
  - Called as `len(strobj)`

Methods

- Associated with a *class* or *type*
- Input comes from arguments ***and*** the string the method was called on
- Example:
  - `strobj.upper()`

How to Use APIs

- Given a problem, break down the problem
  - Can any of the parts of the problem be solved using a method in the API?

Wheel of Fortune

- Determine how many of a certain letter are in a given phrase
- How would we solve this, regardless of case?

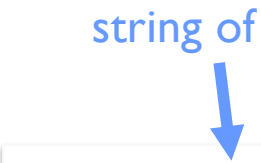
```
def getNumberOfLetters( phrase, letter ):
```

Example Test Cases:

```
test.testEqual( getNumberOfLetters("abracadabra", "a"), 5)
test.testEqual( getNumberOfLetters("Abracadabra", "a"), 5)
test.testEqual( getNumberOfLetters("abracadabra", "A"), 5)
```

Iterating Through a String

- Use a `for` loop to iterate through *characters* in a string

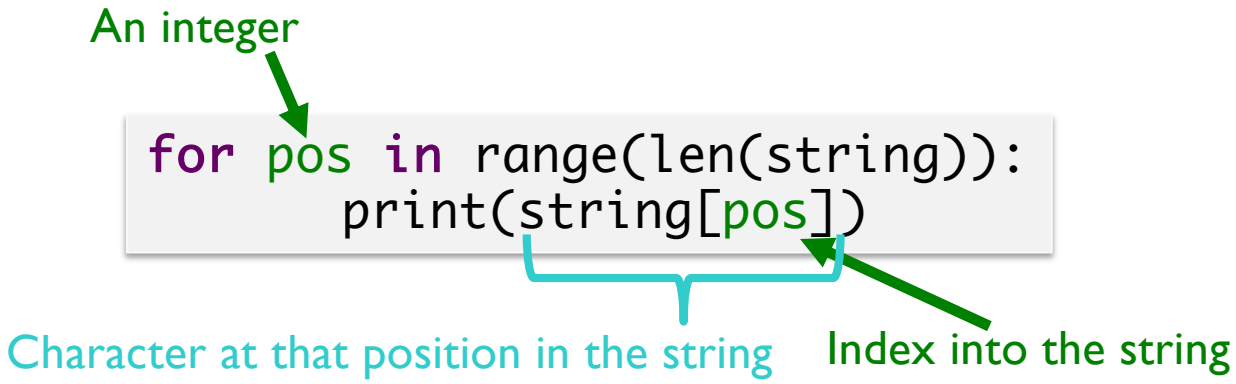



```
for char in string:
    print(char)
```

➢ Read as "for each character in the string"

Iterating Through a String

- Alternatively, can iterate through the *positions* in a string
  - Could write as a **`while`** loop as well



Summary: Iterating Through a String

- For each *character* in the string

string of length 1

```
for char in mystring:
    print(char)
```

Produce the same output!

What comes *after* `in` determines loop's behavior

- For each *position* in the string

An integer

```
for pos in range(len(mystring)):
    print(mystring[pos])
```

Index into the string

Substrings Operator: [:]

- Select a substring (zero or more characters) using the **[ ]** and **:**
  - The values/positions must be integers
- `<sequence>[<start>:<end>]`
  - returns the subsequence from the position **`start`** up to and **not** including **`end`**
- `<sequence>[<start>:]`
  - returns the subsequence from the position **`start`** to the end of the sequence
- `<sequence>[:<end>]`
  - returns the subsequence from the first element up to and **not** including **`end`**
- `<sequence>[:]`
  - returns a copy of the entire sequence

Substrings Operator: [:]

- Select a substring (one or more characters)
- Examples: `filename = "program.py"`

p	r	o	g	r	a	m	.	p	y
0	1	2	3	4	5	6	7	8	9

Expression	Result
`filename[0:2]`	
`filename[0:]`	
`filename[:3]`	
`filename[8:]`	
`filename[-2:]`	

Substrings Operator: [:]

- Select a substring (one or more characters)
- Examples: `filename = "program.py"`

p	r	o	g	r	a	m	.	p	y
0	1	2	3	4	5	6	7	8	9

Expression	Result
`filename[0:2]`	`"pr"`
`filename[0:]`	`"program.py"`
`filename[:3]`	`"pro"`
`filename[8:]`	`"py"`
`filename[-2:]`	`"py"`

String Comparisons

- Same operations as with numbers:
  - ==, !=
  - <, <=
  - >, >=

  } Lexicographical comparison

- Use in conditions, e.g., in `if` statements

```
if course_choice == "CSCI111":
    print("Good choice!")
else:
    print("Maybe next semester")
```

Testing for Substrings

- Using the `in` operator
  - Used `in` before in `for` loops
- Syntax: `substring in string`
  - Evaluates to `True` or `False`
- Example: simple Web search

```
if searchTerm in documentText:
        print(document, "contains", searchTerm)
```

String Search Comparison

- What do the two `if` statements test for?

```
PYTHON_EXT = ".py"

filename = input("Enter a filename: ")

if filename[-(len(PYTHON_EXT)):] == PYTHON_EXT:
        # Appropriate output 1
if PYTHON_EXT in filename:
        # Appropriate output 2
```

Provide some examples for `filename` and state how code would execute

Escape Sequences

- Escape sequences: Represent special characters within a string
  - Considered a single character
- Escape character: \
  - The character following the escape character tells you how to interpret the escape sequence

Escape Sequence	Meaning
\n	Newline character (carriage return)
\t	Tab
\" or \'	Quote
\\	Backslash

demo_str.py

Using Escape Sequences

```
>>> mystr = "Hello!\nHi!"
>>> mystr
'Hello!\nHi!'
>>> print(mystr)
Hello!
Hi!
>>> len(mystr)
10
```

What do these evaluate to?

```
mystr[5]
mystr[6]
mystr[7]
```

Using Escape Sequences

```
>>> mystr = "Hello!\nHi!"
>>> mystr
'Hello!\nHi!'
>>> print(mystr)
Hello!
Hi!
>>> len(mystr)
10
```

What do these evaluate to?

```
mystr[5] → "!"
mystr[6] → "\n"
mystr[7] → "H"
```

Escape Sequences

- Escape sequences: Represent special characters within a string
  - Considered a single character
- Escape character: \
  - The character following the escape character tells you how to interpret the escape sequence
- Example:
  - `print("To print a \\, you must use \"\\\\\"")`
    - What does this display?

Escape Sequence	Meaning
\n	Newline character (carriage return)
\t	Tab
\" or \'	Quote
\\	Backslash

Interactive demonstration

demo_str.py

Review: Description of `print`

- `print(*objects, sep=' ', end='\n', file=sys.stdout)`

Semantics: default values for `sep` is `' '` and `end` is `'\n'`

- Print *object*(s) to the stream *file*, separated by *sep* and followed by *end*.
- Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no *object* is given, *print*() will just write *end*.

https://docs.python.org/3/library/functions.html#print

Practice

- What does this output?

```
print("\n")
```

- Display To print a tab, you must use '\t’.

- Display I said, "How are you?"

Looking Ahead

- Lab 6 Prep due tomorrow
- Lab 6 tomorrow!
- Broader Issue Friday: Autonomous Cars