## Objectives

- Text process, manipulation
  - String operations, processing, methods

## Lab Review

- Banned phrases (in reference to programming/ CS)
  - "Sorry"
    - Unnecessary apologies for *learning*
  - "I don't know"
- Start to learn the questions I ask to figure out issues
- Switch between low level (code itself) and high level (context of program)
- Translation cues (when, becomes)
- Powerful building blocks

## Motivation: Text Processing

- Mostly focused on numbers so far
  - A little on graphics
- We can manipulate strings to do useful work
  - Web search: finding most relevant documents to a query
  - Analyzing web logs (who is looking at my web page?)
  - Many, many others
- **Today's Focus**: the `str` data type and what you can do with them

## Strings: `str`

- Used for text
- Indicated by double quotes "" or single quotes ''
  - In general, I'll use double quotes
  - Empty string: "" or ''
- Use triple quotes """ for strings that go across multiple lines

```
"""This string
is long.
Like, really, really long"""
```

## STRING OPERATIONS

## String Operations

| Operand | Syntax | Meaning |
|---|---|---|
| + | `str1 + str2` | Concatenate two strings into one string |
| * | `str * num` | Concatenate string `num` times |

- Examples:
  - `"I feel " + "sleepy"`
    - Evaluates to `"I feel sleepy"`
  - `"Oops! " * 3`
    - Evaluates to `"Oops! Oops! Oops!"`

Recall lab 0

## More Motivating Constants

- I have a survey program that asks people to rate something on a scale of 1 to 10
- It asks people to rate 100 different things
- I could create the prompt

```
"Rank " + thing + " on a scale of 1 to 10"
```

- But what if my scale changes, and I want it to be on a scale of 1 to 100?
  - I want to make sure the ranking is within my range

## Practice

- Given the following code

```
SCALE_MIN = 1
SCALE_MAX = 10
prompt = …
rating = eval(input( prompt ))
```

- Create the string variable `prompt` for the `input` statement so that it prompts the user:

On a scale of 1 to 10, how much do you like Matt Damon?

survey.py

## String Comparisons

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

} Alphabetical comparison

- Use in conditions in `if` statements

```
if userpick == pick4num:
    print("We have a winner!")
else:
    print("You lose.")
```

string_compare.py

## Strings

- A *sequence* of characters
  - Example:

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

End at `len(band)-1`

characters

| 'T' | 'h' | 'e' | ' ' | 'B' | 'e' | 'a' | 't' | 'l' | 'e' | 's' |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Start at 0

**index** or position of characters

Length of the string: 11

Built-in function: `len(string)` to find length of a string

## Iterating Through a String

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

string of length 1

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

  - Read as "for each character in the string"

Python shell

## 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 shell

## Substrings Operator: []

- Look at a particular character in the string
  - Syntax: `string[<integer_expression>]`
- Examples with `band = “The Beatles”`

| T | h | e |   | B | e | a | t | l | e | s |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| Expression | Result |
|---|---|
| `band[0]` | |
| `band[3]` | |
| `band[len(band)]` | |
| `band[len(band)-1]` | |
| `band[-1]` | |

## Substrings Operator: []

- Look at a particular character in the string
  - Syntax: `string[<integer expression>]`
- Examples with `band = “The Beatles”`

| T | h | e |   | B | e | a | t | l | e | s |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

| Expression | Result |
|---|---|
| `band[0]` | `"T"` |
| `band[3]` | `" "` |
| `band[len(band)]` | `IndexError` |
| `band[len(band)-1]` | `"s"` |
| `band[-1]` | `"s"` |

## Iterating Through a String

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

An integer

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

Index into the string

string_iteration.py

## Summary: Iterating Through a String

- For each character in the string

string of length 1

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

Determines loop’s behavior

- For each position in the string

An integer

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

Index into the string

## Substrings Operator: [:]

- Select a substring (zero or more characters) using the **[ ]** and **:**
- `<sequence>[<start>:<end>]`
  - returns the subsequence from **start** up to and **not** including **end**
- `<sequence>[<start>:]`
  - returns the subsequence from **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) using the [ ] and :
- 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:]` | |
| `filename[0:2]` | |
| `filename[:3]` | |
| `filename[8:]` | |
| `filename[-2:]` | |

## Substrings Operator: [:]

- Select a substring (one or more characters) using the [ ] and :
- 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:]` | `"program.py"` |
| `filename[0:2]` | `"pr"` |
| `filename[:3]` | `"pro"` |
| `filename[8:]` | `"py"` |
| `filename[-2:]` | `"py"` |

## Testing for Substrings

- Using the `in` operator
  - Used `in` before in `for` loops
- Syntax:

```
substring in string:
```

  - Evaluates to `True` or `False`
- Example:

```
if “cat” in name:
    print(name, “contains ‘cat’”)
```

## 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
if PYTHON_EXT in filename:
    # Appropriate output
```

How would the program execution change if it were an **if-elif**?

## Strings are Immutable

You cannot change the value of strings

- For example, you **cannot** change a character in a string
  - ~~str[0] = ‘S’~~

## Revised Pick 4 Game

- To play: pick 4 numbers between 0 and 9
- To win: select the numbers that are selected by the magic ping-pong ball machine

- Done previously: Simulate the magic ping-pong ball machines
- Additional Functionality:
  - Determine if the user picks the winning number

## Looking Ahead

- Friday
  - Lab 3 due
  - Broader Issue: DARPA Urban Challenge
- Exam next Wednesday!
  - Study document up on web site
  - Includes what we’re doing in the next two classes and the next lab

## Revised Pick4 Numbers

- Tell the user how many numbers they got right
  - Get prizes for having some numbers right
- Examples:

| Pick4 Num | User's Pick | Num Correct |
|---|---|---|
| "7737" | "1234" | 1 |
| "0204" | "1234" | 2 |
| "1234" | "1234" | 4 |

pick4num_places.py

## USING THE STR API

## Review

- What is an API?
- How do we call methods on an object?

## `str` Methods

- `str` is a ***class*** or a ***type***
- **Methods**: available operations to perform on `str` objects
  - Provide common functionality
- To see all methods available for `str` class
  - `help(str)`

## `str` Methods

- Example method: **`find(substring)`**
  - Finds the index where substring is in string
  - Returns -1 if substring isn't found
- To call a method:
  - `<str_obj>.methodname([arguments])`
  - Example: `filename.find(".py")`

Executed on this string

## Common `str` Methods

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

string_methods.py

## 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])` | Return 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()` | Return a copy of the string with the leading and trailing whitespace removed |
| `join(<sequence>)` | Return a string which is the concatenation of the strings in the sequence with the string this is called on as the separator |
| `swapcase()` | Return a copy of the string with uppercase characters converted to lowercase and vice versa. |

## String Methods vs. Functions

**Functions**

- All "input" as arguments/ parameters
- Example: `len` is a built-in function
  - Called as `len(strobj)`

**Methods**

- "Input" are argument/ parameters ***and*** the string the method was called on
- Example:
  - `strobj.upper()`

## Are You Smarter Than a 5th Grader?

- Problem in spelling from the show: How many a's are in abracadabra?
  - Solve using `str` methods

## Verifying User Input

- How can we verify that the user entered their lottery number in the correct format?

## Get the Username

- Given the directory formatted as
  - `dir = "/home/www/users/username/"`
- Get the username out

## ESCAPE SEQUENCES

## Escape Sequences

- \ - special character, the ***escape*** character

| Character | Meaning |
|---|---|
| ‘\n’ | New line |
| ‘\t’ | Tab |
| ‘\\’ | Backslash |
| ‘\”’ | Quote |

## Using `str` Methods

- Modify `binaryToDecimal.py` to verify that the entered string contains only numbers
  - Keep asking them for a number until the string contains only numbers

## Using `str` Methods

- Modify `binaryToDecimal.py` to verify that the entered string contains only numbers
  - Keep asking them for a number until the string contains only numbers
- 2nd modification: How could we make sure that entered string contains only 0s and 1s?

## Implementing Wheel of Fortune

- Simplifications: no money, no buying vowels, no keeping track of previous guesses, one player
- Functionality
  - Displaying puzzle appropriately
  - Gets guesses from user
    - Either letters or solve the puzzle
  - Keep track of the number of guesses
  - Displays puzzle with guesses filled in
- Think about …
  - What do we need to model? How would we model it?
  - User input robustness?
  - Any special cases?

`wheeloffortune.py`

## Implementing Wheel of Fortune

- Differences between real and simulated game
  - Players type in letter rather than say it
    - Case matters
    - What if enter more than one letter

## Implementing Wheel of Fortune

- User input verification
  - How can we ensure that the user typed only one letter?
  - How can we ensure that the user typed a *letter*?
- Checking the guess
  - How can we tell if the guessed letter is in the puzzle?
  - How can report the number of times the guessed letter occurs in the puzzle?

## Implementing Wheel of Fortune

- How many times should we prompt the user for a guess?
- How can we display the current puzzle?
  - What does the puzzle look like when we start the game?
  - What does it look like after we correctly guess a letter?

## Wheel of Fortune

- Practice: Modify displayed puzzle to handle punctuation
  - Include punctuation in displayed puzzle
  - Original code:

```
displayedpuzzle = ""
for char in PHRASE:
    if char != " ":
       displayedpuzzle += "_"
    else:
       displayedpuzzle += " "
```

## How Many Numbers Correct?

```
numCorrect = 0
# Don't want to count hyphens, so look at
# every other position, starting at 0
for i in range(0, len(pickedNum), 2):
    if pickedNum[i] == winningNum[i]:
        numCorrect+=1
```

- Why do we have to represent the pickedNum and winningNum as strings?
  - What problem would we run into if we considered them numbers?