ECOR 1041
Computation and Programming

Character Strings and User Input

Carleton University

References

- *Practical Programming*, 3rd ed.
  - Chapter 4, all sections (pp. 65 - 75)
  - Chapter 5, section *Comparing Strings* (pp. 85 - 86)
  - Chapter 3, section *Writing and Running a Program* (pp. 58 - 60)
  - Chapter 9, section *Repetition Based on User Input* (pp. 162 - 165)

Lecture Objectives

- Introduce Python's `str` (character string) type
- Learn to deal with user input
- Provide examples of a simple interactive program

Character Strings

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - Character string, type `str`
  - Escape sequence
  - Concatenation and replication operators
  - Built-in `len`, `str`, `int`, and `float` functions

Learning Outcomes

- Design and implement functions that create string literals and use the operators and built-in functions that operate on strings

Character Strings and Type str

- Many programs are designed to process text: word processors, web browsers, programming-language compilers, etc.
- Programming languages typically store text as a sequence of characters known as a *character string* (often abbreviated to *string*)
- In Python, character strings have type `str`

String Literals

- A string literal consists of characters enclosed by a pair of single quotes, a pair of double quotes, triple single quotes, or triple double quotes
- These 4 strings contain the same sequence of characters:
- `'ECOR 1041'`
- `"ECOR 1041"`
- `'''ECOR 1041'''`
- `"""ECOR 1041"""`

Strings Containing Quotation Marks

- Strings enclosed in single quotes can contain one or more double quotes
- `'Textbook: "Practical Programming, 3rd ed."'`
- Strings enclosed in double quotes can contain one or more single quotes
- `"Textbook: 'Practical Programming, 3rd ed.'"`

Strings Containing Quotation Marks

- We may use an *escape sequence* to represent quote characters in a string
- `'Textbook: \'Practical Programming, 3rd ed.\''`
- `\'` means: Treat the single quote after the backslash as a character in the string, and not the end-of-string delimiter.
- There is also a `\"` escape sequence
- `"Textbook: \"Practical Programming, 3rd ed.\""`

Some Other Escape Sequences

- `\n` represents the *newline* character
- Built-in function `print` displays its arguments on the screen
- `print('Spam!\nSpam!\nSpam!')` displays

```
Spam!
Spam!
Spam!
```

Printing `\n` causes the cursor to move to the start of the next line.

Some Other Escape Sequences

- `\\` represents the backslash character
- Example: to represent this Windows path:

```
D:\courses\ECOR 1041
```

as a Python string, use the `\\` escape sequence to represent each backslash in the path:

```
'D:\\courses\\ECOR 1041'
```

Multiline Strings

- Strings enclosed in triple single quotes or triple double quotes can span multiple lines:
- Example:

```
'''ECOR 1041
ECOR 1042'''
```

  - is equivalent to: `'ECOR 1041\nECOR 1042'`

Multiline Strings

- Example 2:

```
"""ECOR 1041
ECOR 1042"""
```

- is equivalent to: `"ECOR 1041\nECOR 1042"`
- We have used docstrings extensively to document functions
- By convention, a docstring always begins and ends with three double quotes

String Operations - Length

- The built-in `len` function returns the number of characters in a string
- The enclosing quotes are not included in the count of characters, but spaces are

```
>>> len('ECOR 1041')
9
```

- An escape sequence counts as 1 character

```
>>> len('ECOR 1041\nECOR 1042')
19
```

String Operations - Concatenation

- The + operator with string operands is the *concatenation* operator:
- + creates a new string that is the concatenation of the left-hand and right-hand strings

```
>>> 'Hello,' + ' world!'
'Hello, world!'
```

String Operations - Replication

- The `*` operator is the string-replication operator when one operand is an integer *n* and the other operand is a string
- It creates a new string containing *n* copies of the string operand

```
>>> 'Ha' * 3
'HaHaHa'
>>> 2 * 'Bye'
'ByeBye'
```

Iterating Over Strings

- A string is a sequence of characters, so we can use a `for` loop to iterate over every character in a string, starting with the first one:

```
for variable in string:
    block
```

- At the beginning of each iteration, *`variable`* is assigned the next character in the string

Iterating Over Strings

- Example: count the number of spaces in a string

```
s = 'To be, or not to be, that is the question.'
num_spaces = 0
for ch in s:
    if ch == ' ':
        num_spaces += 1
print('Number of spaces:', num_spaces)
```

Type Conversion Functions

- Built-in function `str` returns a string representation of its argument

```
>>> str(42)
'42'
>>> str(3.1415)
'3.1415'
```

Type Conversion Functions

- Built-in function `int` , when passed a string that represents an integer, returns the integer value

```
>>> int('42')
42
>>> int('-123')
-123
>>> int('17.2')
builtins.ValueError: invalid literal for
int() with base 10: '17.2'
```

Type Conversion Functions

- Built-in function `float`, when passed a string that represents a number, returns the numeric value as a `float`

```
>>> float('42')
42.0
>>> float('3.1415')
3.1415
>>> float('XIV')
builtins.ValueError: could not convert
string to float: 'XIV'
```

User Input

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - Built-in `input` function
  - `break` statements
  - `try`, `except`, and `else` blocks
  - f-strings

Learning Outcomes

- Write simple interactive programs that prompt a user for input and process that data
- Write programs that verify if user input is valid and recover from errors

Getting Input From the Keyboard

- Built-in function `input` reads a line of text typed at the keyboard and returns it as a `str`

```
>>> name = input()
Gwen Cooper<Enter>  # chars typed by user
>>> name
'Gwen Cooper'
>>> x = input()
5<Enter> # numeric input is returned as a string
>>> x
'5'
```

A Simple Interactive Program

- *ECOR1041_L10_math_quiz_v1.py* is a simple math drill program
- It asks 10 questions of the form:

  Question i. What is a plus b?

  - *a* and *b* are random integers between 1 and 10, inclusive
- After the user answers a question, feedback is provided
- After all questions are answered, a summary is printed

A Simple Interactive Program

- Random integers are generated by the `randint` function imported from module `random`

```
num1 = random.randint(min_value, max_value)
num2 = random.randint(min_value, max_value)
```

- `randint` returns a random integer between `min_value` and `max_value`, inclusive

A Simple Interactive Program

- Questions are printed and answers are input by this code:

```
prompt = ("Question " + str(question) +
          ". What\'s " + str(num1) +
          " plus " + str(num2) + "? ")
answer = int(input(prompt))
```

- `input` takes an optional string argument (`prompt`), which is displayed before it waits for input to be typed
- `int` is called to convert the response to an integer

Repetition Based on User Input

- Version 1 runs the quiz (calls `ask_questions)` once

```
correct = ask_questions(NUM_QUESTIONS, MIN_VALUE,
          MAX_VALUE)
print("I asked you", NUM_QUESTIONS, "questions.",
        end = " ")
print("You got", correct, "of them right.")
```

- We can change the program to run the quiz repeatedly, based on input from the user (see next slide)

Repetition Based on User Input (*ECOR1041_L10_math_quiz_v2.py*)

```
command = input("Take the quiz or Quit (T, Q): ")
while command != "Q":
    if command == 'T':
        correct = ask_questions(NUM_QUESTIONS, MIN_VALUE,
                                MAX_VALUE)
        print("I asked you", NUM_QUESTIONS, "questions.",
              end = " ")
        print("You got", correct, "of them right.")
    else:  # Command is not Q or T
        print(command, "is not a recognized command")
    command = input("Take the quiz or Quit (T, Q): ")
```

Alternate Form: Repetition Based on User Input (*ECOR1041_L10_ math_quiz_v3.py*)

```
done = False
while not done:
    command = input("Take the quiz or Quit (T, Q): ")
    if command == 'T':
        correct = ask_questions(NUM_QUESTIONS, MIN_VALUE,
                                MAX_VALUE)
        print("I asked you", NUM_QUESTIONS, "questions.",
            end = " ")
        print("You got", correct, "of them right.")
    elif command == 'Q':
        done = True
    else:  # The user typed a command other than Q or T
        print(command, "is not a recognized command")
```

Alternate Form using `break`: Repetition Based on User Input (*ECOR1041_L10_ math_quiz_v4.py*)

```
while True: # loop "forever" (until break)
    command = input("Take the quiz or Quit (T, Q): ")
    if command == 'Q':
        break
    if command == 'T':
        correct = ask_questions(NUM_QUESTIONS, MIN_VALUE,
                                MAX_VALUE)
        print("I asked you", NUM_QUESTIONS, "questions.",
            end = " ")
        print("You got", correct, "of them right.")
    else:  # The user typed a command other than Q or T
        print(command, "is not a recognized command")
```

Alternate Form using `break`: Repetition Based on User Input (*ECOR1041_L10_ math_quiz_v4.py*)

- `break` exits from the loop and continues execution after the end of that loop
- In the case of nested loops, `break` only exits from the loop in which it is coded
- The use of `break` is quite intuitive in loops where we prompt for input but, in general, the use of `break` is not recommended
- **We do not permit the use of `break` in ECOR 1041, but you will use it in ECOR 1044**

Dealing with Invalid User Input

- In the first four versions of our program, what happens if the user enters an answer that is not an integer?
- Example 1:

```
Question 1. What's 4 plus 4? 8.5
builtins.ValueError: invalid literal for
int() with base 10: '8.5'
```

- Example 2:

```
Question 2. What's 6 plus 2? x
builtins.ValueError: invalid literal for
int() with base 10: 'x'
```

Dealing with Invalid User Input: `try` and `except`

- How can we deal with this “nicely”?
- Python provides `try` and `except` blocks to recover from errors and continue the program
- Example:

```
try:
    answer = int(input(prompt))
except:
    print("An error occurred")
```

Dealing with Invalid User Input: `try` and `except`

- If an error occurs in the `try` block, the `except` block is executed
- If no error occurs in the `try` block, the `except` block is not executed

```
try:
    answer = int(input(prompt))
except:
    print("An error occurred")
```

Dealing with Invalid User Input: `try`, `except`, `else`

- We may also include an `else` block
- The `else` block is executed if no error occurs in the `try` block

```
try:
    answer = int(input(prompt))
except:
    print("An error occurred")
else:
    print("No errors encountered")
```

Dealing with Invalid User Input: `try`, `except`, `else`

- How can we “bullet proof” our math quiz code?
- We could do this:

```
try:
    answer = int(input(prompt))
except:
    print("No, I\'m afraid the answer is",
correct_result)
# else on next slide
```

Dealing with Invalid User Input: `try`, `except`, `else`


```
else:
    if answer == correct_result:
        print("That\'s right -- well done.")
        ok += 1
    else:
        print("No, I\'m afraid the answer
is", correct_result)
```

- We cannot use `elif` here. Why not?

Dealing with Invalid User Input (*ECOR1041_L10_math_quiz_v5.py*)

- We can also do this in a more concise way
- If the user enters invalid input, set `answer` to an incorrect result
- Then we only need the `try` and `except` blocks (not the `else`)

```
try:
    answer = int(input(prompt))
except:
    answer = correct_result + 1
```

Python String Formatting

- Instead of prompting the user as we have been doing:

```
prompt = ("Question " + str(question) +
    ". What\'s " + str(num1) + " plus " +
    str(num2) + "? ")
```

we can use Python String formatting

- There are several different options: https://realpython.com/python-string-formatting/
- We are going to use Version 3, which is best for Python 3.6 + when we are not using user-supplied format strings

Python String Formatting: f-Strings (*ECOR1041_L10_ math_quiz_v6.py*)

- Version 3 is formatted string literals or f-strings
- An f-string starts with the character `f` and is followed by a string, e.g. `f'…'` or `f"…"` or `f'''…'''` or `f"""…"""`
- Any variables included in the f-string are enclosed in braces: `{…}`
- Using f-strings our prompt becomes:

```
prompt = f'Question {question}. What\'s
{num1} plus {num2}? '
```

Recap of Learning Outcomes

Carleton University

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - Character string, type `str`
  - Escape sequence
  - Concatenation and replication operators
  - Built-in `len`, `str`, `int`, and `float` functions

Learning Outcomes

- Be able to design and implement functions that create string literals and use the operators and built-in functions that operate on strings

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - Built-in `input` function
  - `break` statements
  - `try`, `except`, and `else` blocks
  - f-strings

Learning Outcomes

- Write simple interactive programs that prompt a user for input and process that data
- Write programs that verify if user input is valid and recover from errors