

CMSC201

Computer Science I for Majors

Lecture 07 – Strings and Lists

Prof. Katherine Gibson

Last Class We Covered

- One-way, two-way, and multi-way decision structures
 - **if**, **if-else**, and **if-elif-else** statements
- Control structures (review)
- Conditional operators (review)
- Boolean data type (review)
- Coding algorithms using decision structures

Any Questions from Last Time?

Today's Objectives

- To discuss the usage of `eval()` and the potential security concerns
- To learn about lists and what they are used for
- To better understand the string data type
 - Learn how they are represented
 - Learn about and use some of their built-in functions
- To be able to apply string formatting to produce attractive, informative program output

About `eval()`

Previous Uses of `eval()`

- Remember our temperature converter?

What does `eval` do?



```
def main():
    celsius = eval(input("What is the Celsius temperature? "))
    fahrenheit = 9/5 * celsius + 32

    print("The temperature is ", fahrenheit,
          " degrees Fahrenheit.")

main()
```

The Problem with `eval()`

- `eval()` interprets a string as code
- It lets a Python program run Python code within itself
- In our example, we use it to let Python decide what data type to store the input as
 - If the user gives us an integer, store it as an int
 - If the user gives us a decimal, store it as a float
- Using `eval()` is a security hole.

The Problem with `eval()`


- **But** if the user gives us a malicious command to delete files or folders, it may also run that
- If you have `os` imported, and you ask for input **using** `eval(input())`, someone could type malicious code like in response
 - `os.system('rm hw1.py')`
 - This would delete your `hw1.py` file!

What to Do Instead?

- Instead of using `eval()` to cast strings...
- Use the exact type you want to cast to:
 - `int(input())`
 - `float(input())`

Fixing the Temperature Converter

Changed to a float cast



```
def main():  
    celsius = float(input("What is the Celsius temperature? "))  
    fahrenheit = 9/5 * celsius + 32  
  
    print("The temperature is ", fahrenheit,  
          " degrees Fahrenheit.")  
  
main()
```

Introduction to Lists

Exercise: Average Three Numbers

- Read in three numbers and average them

```
num1 = int(input("Please enter a number: "))  
num2 = int(input("Please enter a number: "))  
num3 = int(input("Please enter a number: "))  
print((num1 + num2 + num3) / 3)
```

- Easy! But what if we want to do 100 numbers? Or 1000 numbers?
- Do we want to make 100 or 1000 variables?

Using Lists

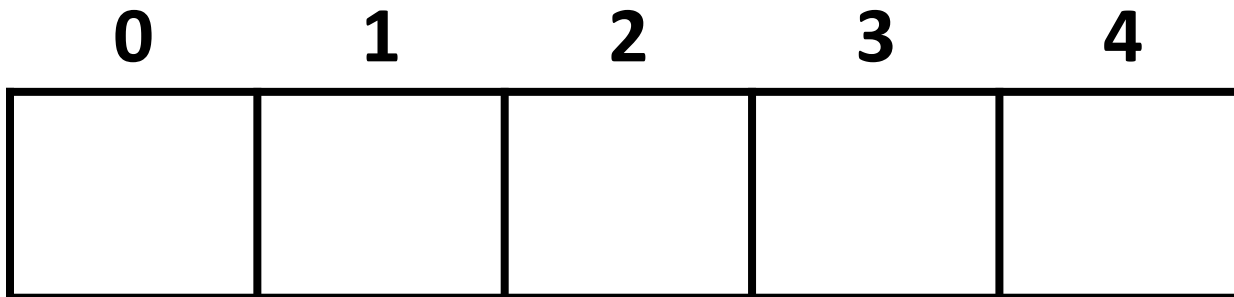
- Need an easy way to hold onto individual data items without needing to make lots of variables
 - Making `num1`, `num2`, ..., `num99`, `num100` is time-consuming and impractical
- Instead, we can use a *list* to hold our data
 - A list is a *data structure*: something that holds multiple pieces of data in one structure

Using Lists: Individual Variables

- We need an easy way to refer to each individual variable in our list
 - Math uses subscripts (x_1, x_2, x_3 , etc.)
 - Instructions use numbers (“Step 1: Combine...”)
- Programming languages use a different syntax
 - `x[1]`, `x[0]`, `instructions[1]`, `point[i]`

Numbering in Lists

- Lists don't start counting from 1
 - They start counting from 0!
- Lists with n elements are numbered from 0 to $n-1$
 - The list below has 5 elements, and is numbered from 0 to 4



Properties of a List

- Heterogeneous (any data type!)
- Contiguous (all together in memory)
- Ordered (numbered from 0 to n-1)

- Have random (instant) access to any element
- Add elements using the append method
- They're "mutable sequences of arbitrary objects"

List Syntax

- Use `[]` to assign initial values (*initialization*)

```
myList = [1, 3, 5]
```

```
words = ["Hello", "to", "you"]
```

- And to refer to individual elements of a list

```
>>> print(words[0])
```

```
Hello
```

```
>>> myList[0] = 2
```

List Example: Grocery List

- You are getting ready to head to the grocery store to get some much needed food
- In order to organize your trip and to reduce the number of impulse buys, you decide to make a grocery list

List Example: Grocery List

- Inputs:
 - 3 items for grocery list
- Process:
 - Store grocery list using list data structure
- Output:
 - Grocery list

Grocery List Code

```
def main():  
    print("Welcome to the Grocery Manager 1.0")  
    // initialize the value and the size of our list  
    grocery_list = [None]*3  
  
    grocery_list[0] = input("Please enter your first item: ")  
    grocery_list[1] = input("Please enter your second item: ")  
    grocery_list[2] = input("Please enter your third item: ")  
    print(grocery_list[0])  
    print(grocery_list[1])  
    print(grocery_list[2])  
main()
```

Grocery List Demonstration

- Here's a demonstration of what the code is doing

```
bash-4.1$ python groceries.py
```

```
Please enter your first item: milk
```

```
Please enter your second item: eggs
```

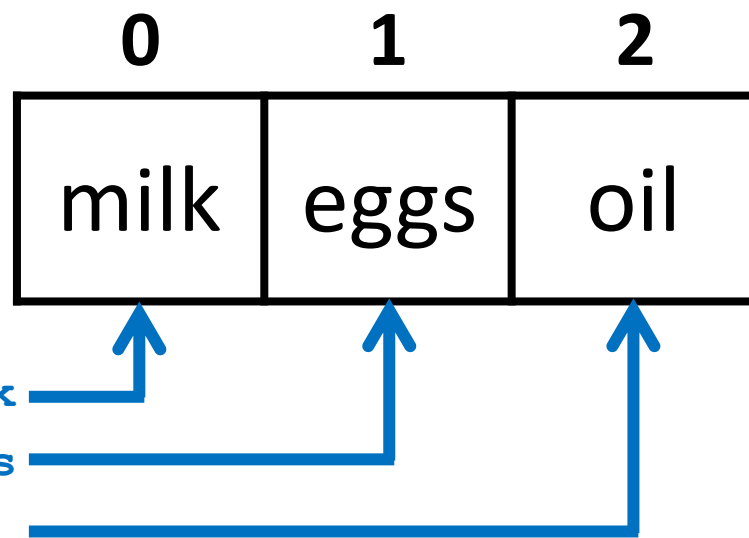
```
Please enter your third item: oil
```

```
milk
```

```
eggs
```

```
oil
```

```
grocery_list[0] = input("Please enter ...: ")
grocery_list[1] = input("Please enter ...: ")
grocery_list[2] = input("Please enter ...: ")
print(grocery_list[0])
print(grocery_list[1])
print(grocery_list[2])
```



List Example: Grocery List

- What would make this process easier?
- Loops!
 - Instead of asking for each item individually, we could keep adding items to the list until we wanted to stop (or the list was “full”)
- We will learn more about loops in the next couple of classes

Strings

The String Data Type

- Text is represented in programs by the string data type
- A *string* is a sequence of characters enclosed within quotation marks (") or apostrophes (')
 - Sometimes called double quotes or single quotes
- *FUN FACT!* – *The most common use of personal computers is word processing*

String Examples

```
>>> str1 = "Hello"  
>>> str2 = 'spam'  
>>> print(str1, str2)  
Hello spam  
>>> type(str1)  
<class 'str'>  
>>> type(str2)  
<class 'str'>
```

Getting Strings as Input

- Using `input()` automatically gets a string

```
>>> firstName = input("Please enter your name: ")
Please enter your name: Shakira
>>> print("Hello", firstName)
Hello Shakira
>>> type(firstName)
<class 'str'>
>>> print(firstName, firstName)
Shakira Shakira
```

Accessing Individual Characters

- We can access the individual characters in a string through *indexing*
- The characters in a string are numbered starting from the left, beginning with 0
 - Does that remind you of anything?

Syntax of Accessing Characters

- The general form is

STRING [EXPR]

- Where **STRING** is the name of the string variable and **EXPR** determines which character is selected from the string

Example String

0	1	2	3	4	5	6	7	8
H	e	l	l	o		B	o	b

```
>>> greet = "Hello Bob"
>>> greet[0]
'H'
>>> print(greet[0], greet[2], greet[4])
H l o
>>> x = 8
>>> print(greet[x - 2])
B
```

Example String

0	1	2	3	4	5	6	7	8
H	e	l	l	o		B	o	b

- In a string of n characters, the last character is at position $n-1$ since we start counting with 0
- Index from the right side using negative indexes

```
>>> greet[-1]
```

```
'b'
```

```
>>> greet[-3]
```

```
'B'
```

Substrings and Slicing

Substrings

- Indexing only returns a single character from the entire string
- We can access a *substring* using a process called *slicing*
 - Substring: a (sub)part of another string
 - Slicing: we are slicing off a portion of the string

Slicing Syntax

- The general form is

STRING [START : END]

- **START** and **END** must both be integers
 - The substring begins at index **START**
 - The substring ends before index **END**
 - The letter at index **END** is not included

Slicing Examples

0	1	2	3	4	5	6	7	8
H	e	l	l	o		B	o	b

```
>>> greet[0:3]
```

```
'Hel'
```

```
>>> greet[5:9]
```

```
' Bob'
```

```
>>> greet[:5]
```

```
'Hello'
```

```
>>> greet[1:]
```

```
'ello Bob'
```

```
>>> greet[:]
```

```
'Hello Bob'
```

Specifics of Slicing

- If **START** or **END** are missing, then the start or the end of the string are used instead
- The index of **END** must come after the index of **START**
 - What would the substring `greet[1:1]` be?
 ' '
 - An empty string!

More Slicing Examples

0	1	2	3	4	5	6	7	8
H	e	l	l	o		B	o	b
-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> greet[2:-3]
```

```
'llo '
```

```
>>> greet[-6:-2]
```

```
'lo B'
```

```
>>> greet[-6:6]
```

```
'llo '
```

```
>>> greet[-9:8]
```

```
'Hello Bo'
```

Forming New Strings - Concatenation

- We can put two or more strings together to form a longer string
- *Concatenation* “glues” two strings together

```
>>> "Peanut Butter" + "Jelly"  
'Peanut ButterJelly'  
>>> "Peanut Butter" + " & " + "Jelly"  
'Peanut Butter & Jelly'
```

Forming New Strings - Repetition

- Concatenating the same string together multiple times can be done with *repetition*
 - Which operator would you use for this?

```
>>> animal = "dogs"
```

```
>>> animal*3
```

```
'dogsdogsdogs'
```

```
>>> animal*8
```

```
'dogsdogsdogsdogsdogsdogsdogsdogsdogs'
```


Length of a String

- To get the length of a string, use `len()`

```
>>> title = "CMSC 201"
```

```
>>> len(title)
```

```
8
```

```
>>> len("Help I'm trapped in here!")
```

```
24
```

- Why would we need the length of a string?

String Operators in Python

Operator	Meaning
<code>+</code>	Concatenation
<code>*</code>	Repetition
<code>STRING[#]</code>	Indexing
<code>STRING[#:#]</code>	Slicing
<code>len(STRING)</code>	Length
<code>for VAR in STRING</code>	Iteration



We'll cover this next class, when we learn **for** loops!

Just a Bit More on Strings

- Python has many, many ways to interact with strings, and we will cover them in detail soon
- For now, here are two very useful functions:
 - `s.lower()` – copy of `s` in all lowercase letters
 - `s.upper()` – copy of `s` in all uppercase letters
- Why would we need to use these?
 - Remember, Python is case-sensitive!

String Processing Examples

Example: Creating Usernames

- Our rules for creating a username:
 - First initial, first 7 characters of last name (lowercase)

```
# get user's first and last names
first = input("Please enter your first name: ")
last  = input("Please enter your last name:  ")

# concatenate first initial with 7 chars of last name
uname = first[0].lower() + last[:7].lower()
print("Your username is: ", uname)
```



Why is this 7?

Example: Creating Usernames

```
>>> first = input("Please enter your first name: ")  
Please enter your first name: Donna  
>>> last = input("Please enter your last name: ")  
Please enter your last name: Rostenkowski
```

```
>>> uname = first[0] + last[:7]  
>>> print("Your username is: ", uname)
```

Your username is DRostenk

Usernames must be lowercase!

```
>>> uname = first[0].lower() + last[:7].lower()  
>>> print("Your username is: ", uname)
```

Your username is drostenk

Example: Creating Usernames

```
>>> first = input("Please enter your first name: ")
```

```
Please enter your first name: Barack
```

```
>>> last = input("Please enter your last name: ")
```

```
Please enter your last name: Obama
```

```
>>> uname = first[0].lower() + last[:7].lower()
```

```
>>> print("Your username is: ", uname)
```

```
Your username is bobama
```

- What would happen if we did `last[7]`?
 - **IndexError** – but why does `last[:7]` work?

Example: Printing the Months

- Given an integer (from 1 to 12) print the three letter abbreviation for that month
- Start by storing all the names in one big string:
`months = "JanFebMarAprMayJunJulAugSepOctNovDec"`
- Use the number of the month to get the right “slice” of the `months` string

Example: Printing the Months

- Let's figure out the position for each month name:

```
months = "JanFebMarAprMayJunJulAugSepOctNovDec"
          0123456789          5          5          5          5
```

Month	Jan	Feb	Mar	Apr	May	Jun
Num	1	2	3	4	5	6
Pos	0	3	6	9	12	15

Month	Jul	Aug	Sep	Oct	Nov	Dec
Num	7	8	9	10	11	12
Pos	18	21	24	27	30	33

Example: Printing the Months

- Notice a pattern?

Month	Jan	Feb	Mar	Apr
Num	1	2	3	4
Pos	0	3	6	9

- To get the position, subtract 1 from the month's number and multiply by 3

$$\text{pos} = (\text{num} - 1) * 3$$

- Use it to get the month name from the string

Example: Printing the Months

```
def main():
    months = "JanFebMarAprMayJunJulAugSepOctNovDec"

    n = int(input("Enter a month number (1-12): "))

    # compute starting position of month n in months
    pos = (n-1) * 3

    # grab the appropriate slice from months
    monthAbbrev = months[pos:pos+3]

    # print the result
    print ("The month abbreviation is", monthAbbrev)

main()
```

Example: Printing the Months

```
bash-4.1$ python months.py
Enter a month number (1-12): 1
The month abbreviation is Jan
```

```
bash-4.1$ python months.py
Enter a month number (1-12): 12
The month abbreviation is Dec
```

```
bash-4.1$ python months.py
Enter a month number (1-12): 100
The month abbreviation is
```

What
happened?

months [297:300]

There's nothing
there in the string!

Announcements

- Your Lab 4 is meeting normally this week!
 - Make sure you attend your correct section
- Homework 3 is out
 - Due by Thursday (Sept 24th) at 8:59:59 PM
- Homeworks are on Blackboard
 - Weekly Agendas are also on Blackboard