Lecture 5: Lists, Strings, and Dictionaries

CS 1109 Summer 2024

Lists

Motivation

- Keep track of how many points you score each game this season in basketball
- You could create a variable for each game
  - Too much repeated code
  - What if we don't know in advance how many games we want to track?

```
game_1 = 10
game_2 = 8
game_3 = 15
game_4 = 11
...
game_82 = 17
```

Lists

- Lists are ordered sets of elements (often called “arrays”)
- Can grow larger if necessary

```
point_totals = [10, 8, 15, 11, ..., 17]
```

Lists

- Lists are ordered sets of elements (often called “arrays”)
- Can grow larger if necessary
- Elements can be any type
    - Even mix and match types

Lists

- Lists are ordered sets of elements (often called “arrays”)
- Can grow larger if necessary
- Elements can be any type
  - Even mix and match types

```
x = [8, “guitar”, True]
```

Lists

- Lists are ordered sets of elements (often called “arrays”)
- Can grow larger if necessary
- Elements can be any type
  - Even mix and match types
- To add to the end of a list use `.append()`
  - `x.append(7)`

```
x = [8, “guitar”, True]
```

Lists

- Lists are ordered sets of elements (often called “arrays”)
- Can grow larger if necessary
- Elements can be any type
  - Even mix and match types
- To add to the end of a list use `.append()`
  - `x.append(7)`
- How do you add to the beginning of a list?
- How do you access elements?

```
x = [8, “guitar”, True]
```

Counting in CS

- How do we normally count?

Counting in CS

- How do we normally count?

Counting in CS

- How do we normally count?

Counting in CS

- How do we normally count?
- Student class
  - Freshman - have completed zero years

Counting in CS

- Zero-based numbering is very common in CS
  - “Repeat 5 times” - 0, 1, 2, 3, 4
- Detour into common C language for loop
  - `for(i=0; i<5; i++)` vs. `for(i=1; i<6; i++)` vs. `for(i=0; i<=4; i++)`
  - Zero based numbering here makes it easy to see how many times loop executes
- Python uses zero-based numbering
  - `range(3)` gives 0, 1, 2

Counting in CS - Why use zero-based numbering?*

- Resource constraints?
  - Assign 10 students an ID number, but only use a single digit

Counting in CS - Why use zero-based numbering?*

- Resource constraints?
  - Assign 10 students an ID number, but only use a single digit
- Memory address logic?
  - A contiguous list stored at address *p*:
    - Element addresses should be *p + o*, where *o* is the offset
    - First element at offset 0

Counting in CS - Why use zero-based numbering?*

- Resource constraints?
  - Assign 10 students an ID number, but only use a single digit
- Memory address logic?
  - A contiguous list stored at address *p*:
    - Element addresses should be *p + o*, where *o* is the offset
    - First element at offset 0
- Benefits complex algorithms?
  - Anything with modulo operator

Counting in CS - Why use zero-based numbering?*

- Resource constraints?
  - Assign 10 students an ID number, but only use a single digit
- Memory address logic?
  - A contiguous list stored at address *p*:
    - Element addresses should be *p + o*, where *o* is the offset
    - First element at offset 0
- Benefits complex algorithms?
  - Anything with modulo operator
- Edsger Dijkstra argument for zero-based numbering?

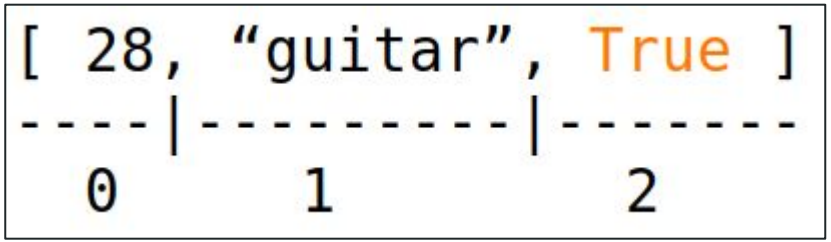
* See here for more discussion

How does that relate to lists?

- Used widely in programming languages with lists (arrays)
  - Called zero-based indexing when used for lists
- First element is stored at index 0, second at index 1, etc.

How does that relate to lists?

- Used widely in programming languages with lists (arrays)
  - Called zero-based indexing when used for lists
- First element is stored at index 0, second at index 1, etc.



Back to Lists

- How do you access items in a list?
  - `x[<index>]`
  - `x[0]` for first item in the list
- Examples
  - `print(x[1])`
  - `x[6] = 30`
  - `x[4] = x[0]`

Back to Lists

- How do you access items in a list?
  - `x[<index>]`
  - `x[0]` for first item in the list
- Examples
  - `print(x[1])`
  - `x[6] = 30`
  - `x[4] = x[0]`
- `len(x)` gives you length/size of list

Back to Lists

- How do you access items in a list?
  - `x[<index>]`
  - `x[0]` for first item in the list
- Examples
  - `print(x[1])`
  - `x[6] = 30`
  - `x[4] = x[0]`
- `len(x)` gives you length/size of list
- How would you access the last element of a list?

Back to Lists

- How do you access items in a list?
  - `x[<index>]`
  - `x[0]` for first item in the list
- Examples
  - `print(x[1])`
  - `x[6] = 30`
  - `x[4] = x[0]`
- `len(x)` gives you length/size of list
- How would you access the last element of a list?
  - `x[len(x)-1]`
    - Without `-1`, index out of range error

Back to Lists

- Add items to the back of a list
  - `.append(<element>)`
- How do you add to the front of a list?
  - `.insert(<index>, <element>)`
  - `x.insert(0, 23)` to add 23 at index 0 of the list
- How do I add to a middle index?

Back to Lists

- Add items to the back of a list
  - `.append(<element>)`
- How do you add to the front of a list?
  - `.insert(<index>, <element>)`
  - `x.insert(0, 23)` to add 23 at index 0 of the list
- How do I add to a middle index?
  - Still `.insert(<index>, <element>)`
  - `x.insert(7, 8)` to insert 8 at as 8th item

Lists

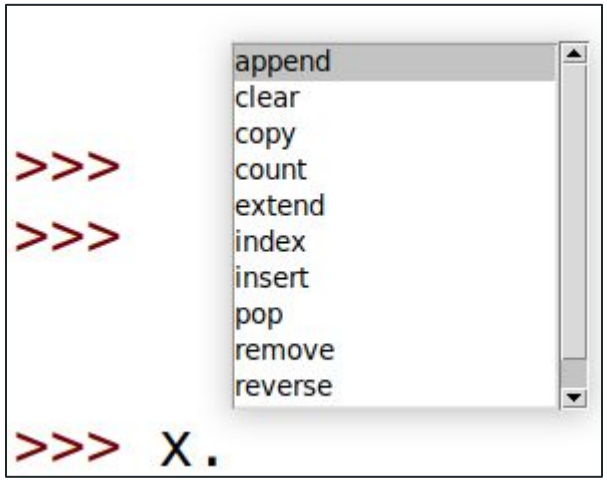
- Slices - subsets of a list
- To get a slice, use
  - `x[<start>: <end>]` , where `end` is not inclusive
- Leave `end` empty to take every element after `start`
  - `x[2:]`
- Leave `` `start` `` empty to take every element before `end`
  - `x[:2]`

Lists

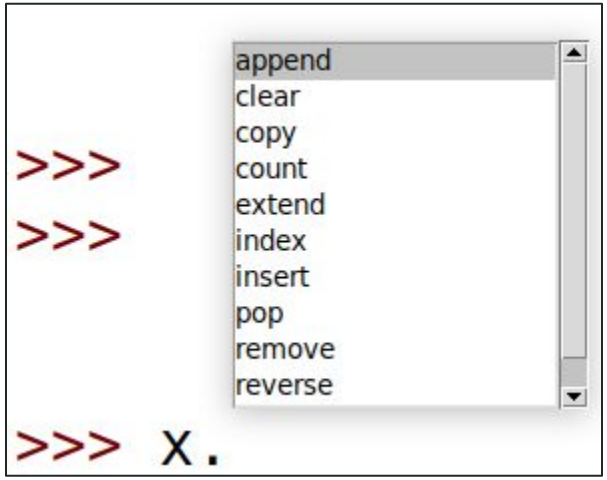
- Deletion - Python has two ways
- `x.remove(<value>)` deletes the first instance of value in list
- `del(x[2])` deletes element at index 2

Lists

- Other useful list operations
  - `x + y` - plus operator creates one list with all items
  - `x.sort()` - sort list with only single value type
  - `x.reverse()` - reverse order of list
- In the IDLE shell, pressing tab after typing the period to show options

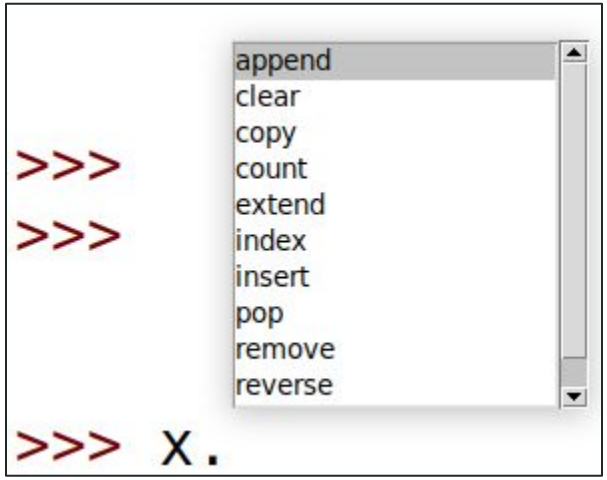


Lists



- This is an approved form of autocomplete
- Use it to explore operations for different types

Lists



- This is an approved form of autocomplete
- Use it to explore operations for different types
- Check with the Python docs to see details about various operations

Strings

Strings vs. Lists

- Strings are similar to lists
- Both are ordered sets (sequences)
  - Strings can only contain characters
- Both can access each individual element with brackets
  - `x = “Beautiful”`
  - `print(x[3])`
  - `y = x[0:5]`
- Share some operations
  - `len()`
  - Concatenation (**+**) works for both
- Both iterable (discussed later)

Strings vs. Lists

- Strings are immutable
  - Cannot change or delete characters once created
  - Must be recreated completely
- Lists are mutable
  - Elements can be changed/deleted
- Most operations differ
- Built-in string operations tend to return string
  - Leave original string in tact
  - Must store in new variable for modification to persist
  - `y = x.upper()`

Useful String Operations*

- `.upper()` - makes all characters upper case
  - `.lower()` - makes lower case
- `.count(<substring>)` - counts number of times substring appears
- `.find(<substring>)` - returns index matching first character of substring
- `.replace(<old>, <new>)` - replaces old substring with new
- `.strip()` - removes beginning and ending whitespace

* Check with Python docs for more

Sequences

- A sequence is an ordered set of items
- Strings and lists are types of sequences
- Most sequences have common operations (see here)
- Sequences can be iterated through
- E.g., `range()` returns a sequence that we can loop through

Looping through sequences

```
for i in range(1,11):
    print(i)
```

```
food = "lasagna"
for letter in food:
    print(letter)
```

```
numbers = [1,2,3,4,5]
for num in numbers:
    print(num)
```

Looping through sequences

Iterate with each element

```
food = "lasagna"
for letter in food:
    print(letter)
```

```
numbers = [1,2,3,4,5]
for num in numbers:
    print(num)
```

Iterate with indices

```
food = "lasagna"
for i in range(len(food)):
    print(food[i])
```

```
numbers = [1,2,3,4,5]
for i in range(len(numbers)):
    print(numbers[i])
```

Looping through sequences

- Be careful with trying to delete from lists using loops
- Python might forgive for doing it when looping with elements
- But delete item using index looping and get an error
  - Why?

```
>>> x = [1,2,3,4]
>>> for i in range(len(x)):
        if x[i] == 1:
                del(x[i])
Traceback (most recent call last):
  File "/usr/lib/python3.8/idlelib/run.py", line 559, in runcode
    exec(code, self.locals)
  File "<pyshell#91>", line 2, in <module>
IndexError: list index out of range
```

Looping through sequences

- Use the `in` keyword to check if item in list
- Then just delete

```
>>> x = [1,2,3,4]
>>> if 1 in x:
        x.remove(1)


>>> x
[2, 3, 4]
```

Dictionaries

Problem

- How would you store a list of books and their authors?
  - Not immediately clear with the tools we have
- Want to look up author using book title

Problem

- How would you store a list of books and their authors?
  - Not immediately clear with the tools we have
- Want to look up author using book title

```
books_and_authors = ["Pride and Prejudice","Jane Austen", \
                     "Moby Dick", "Herman Melville" ]
```

```
books = ["Pride and Prejudice", "Moby Dick"]
authors = ["Jane Austen", "Herman Melville"]
```

Dictionaries

- English dictionary look up key (a word), get a value (definition)
  - One-way - normally, can’t look up definition to find a word

Dictionaries

- English dictionary look up key (a word), get a value (definition)
  - One-way - normally, can’t look up definition to find a word
- Python dictionaries do the same thing
- Key is book title; value is author

Dictionaries

- English dictionary look up key (a word), get a value (definition)
  - One-way - normally, can't look up definition to find a word
- Python dictionaries do the same thing
- Key is book title; value is author

```
books = {"Pride and Prejudice":"Jane Austen", \
         "Moby Dick": "Herman Melville"}
```

Dictionaries

- English dictionary look up key (a word), get a value (definition)
  - One-way - normally, can't look up definition to find a word
- Python dictionaries do the same thing
- Key is book title; value is author
- `Dictionary = {<key>: <value>, <key>: <value>}`

```
books = {"Pride and Prejudice":"Jane Austen", \
         "Moby Dick": "Herman Melville"}
```

Dictionaries

- English dictionary look up key (a word), get a value (definition)
  - One-way - normally, can't look up definition to find a word
- Python dictionaries do the same thing
- Key is book title; value is author
- `Dictionary = {<key>: <value>, <key>: <value>}`
- What if we wanted to lookup a book based on author?

```
books = {"Pride and Prejudice":"Jane Austen", \
         "Moby Dick": "Herman Melville"}
```

Dictionaries

```
books = {"Pride and Prejudice":"Jane Austen", \
         "Moby Dick": "Herman Melville"}
```

```
authors = {"Jane Austen":"Pride and Prejudice", \
           "Herman": "Moby Dick"}
```

Dictionaries

- Looping through dictionaries gives keys only
- Access values with bracket notation, but use key instead of index

Dictionaries

- Looping through dictionaries gives keys only
- Access values with bracket notation, but use key instead of index

```
for author in authors:
    print(author)
    print(authors[author])
```

```
Herman Melville
Moby Dick
Jane Austen
Pride and Prejudice
```

Dictionaries

- Do not try to access dictionaries like lists with indices

```
>>> for i in range(len(authors)):
        print(authors[i])


Traceback (most recent call last):
  File "/usr/lib/python3.8/idlelib/run.py", line 559, in runcode
    exec(code, self.locals)
  File "<pyshell#155>", line 2, in <module>
KeyError: 0
```

Dictionaries

- It is possible to find key from values

```
>>> if "Moby Dick" in authors.values():
        for author in authors:
            if authors[author] == "Moby Dick":
                print(author)


Herman Melville
```

Dictionaries

- Many times helpful to name dictionary after key and value
  - `author_to_book` - convert an author to a book
  - `book_to_author` - convert a book to its author

Dictionaries

- Many times helpful to name dictionary after key and value
  - `author_to_book` - convert an author to a book
  - `book_to_author` - convert a book to its author

```
book_to_author = {"Pride and Prejudice":"Jane Austen", \
          "Moby Dick": "Herman Melville"}

author_to_book = {"Jane Austen":"Pride and Prejudice", \
            "Herman": "Moby Dick"}

for auth in author_to_book:
    print(author_to_book[auth])
```

Dictionaries

- To add an item, use bracket notation:
  - `author_to_book[“George Orwell”]  = “1984”`
- To change item, same thing:
  - `author_to_book[“George Orwell”] = “Animal Farm”`
- To delete:
  - `del(author_to_book[“George Orwell”])`