CSCI 1101 - COMPUTER SCIENCE 1

Lists
Lists

• A list is a sequence of values of any data-type enclosed in brackets [].

• Examples:
  • [10, 20, 25, 30, 35, 40, 45, 50]
  • ['blue', 'green', 'red', 'yellow']
  • [1.0, 1.5, 2.0, 2.5, 3.0]

I can even mix data-types in the same list.
• [10, -90, 10.54, “green”]

I can even have a list within another list (nested lists)
• [10, 20, ['blue', 'green', 'pink'], [3.0, 7.0, 5.6 ] ]
• Can assign a list to a variable

```python
```

• Accessing the elements of a list by specifying the index inside brackets… similar to string!

```python
>>> cheese [0]    # returns element at index 0
Cheddar
>>> cheese [1:4]  # returns elements from index 1 to 3
Swiss, Havarti, American
>>> cheese [3:]   # returns elements from index 3 to end of string
American, Provolone
>>> cheese [:2]   # returns elements from index 0 to 1
Cheddar, Swiss
```
Lists (Cont.)

- Can join two lists using the + operator

```python
>>> cheese = ['Cheddar', 'Swiss', 'Provolone']
>>> meat = ['Ham', 'Pepperoni']
>>> toppings = cheese + meat
>>> print(toppings)
['Cheddar', 'Swiss', 'Provolone', 'Ham', 'Pepperoni']
```
Lists (Cont.)

- Can **repeat** elements in a list using the * operator

```python
>>> cheese = [ 'Cheddar', 'Swiss', 'Provolone' ]
>>> print (cheese * 2)
[ 'Cheddar', 'Swiss', 'Provolone', 'Cheddar', 'Swiss', 'Provolone' ]

>>> sequence = [ 1, 2, 3, 4, 5 ]
>>> print (sequence * 3)
[ 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5 ]
```
Lists (Cont.)

- Can update elements (or items) in the list

```python
>>> cheese = ['Cheddar', 'Swiss', 'Havarti', 'American', 'Provolone']

>>> cheese[1] = "Low-Fat Swiss"

>>> cheese[:2] = ['Sharpe Cheddar', 'Baby Swiss']

Index 0 and 1, so update first two elements in the list
```
Lists (Cont.)

- Can update elements (or items) in the list

```
cheese = ['Cheddar', 'Swiss', 'Havarti', 'American', 'Provolone']
```
```
cheese[:2] = ['2% Swiss', 'Baby Swiss', 'Mild Swiss']
```

Index range is only two, but the list contains 3 elements… what happens?

```
print(cheese)
```
```
['2% Swiss', 'Baby Swiss', 'Mild Swiss', 'Havarti', 'American', 'Provolone']
```
Lists (Cont.)

• Can update elements (or items) in the list

```
>>> cheese = ['Cheddar', 'Swiss', 'Havarti', 'American', 'Provolone']
>>> cheese[1] = "Low-Fat Swiss"
>>> cheese[:2] = ['Sharp Cheddar', 'Baby Swiss']
```

Index 0 and 1, so update first two elements in the list
Lists (Cont.)

- Defining a list...

```python
>>> cheese = list()
>>> type(cheese)
<type 'list'>
```

- Appending to list (adds to the end of the list)

```python
>>> cheese.append('Swiss')
['Swiss']
>>> cheese.append('Cheddar')
['Swiss', 'Cheddar']
```

- Remove element from the list

```python
>>> cheese.remove('Swiss')
['Cheddar']
```

The function `append (x)`, will add the value `x` to the end of the list.

The function `remove (x)`, removes the element `x` from list if it exists. If `x` is not in the list, an error is printed.
Lists (Cont.)

• Inserting an element at a given index

```python
>>> seasons = [ 'Fall', 'Spring' ]
>>> seasons.insert( 1 , 'Winter' )
>>> print(seasons)
[ 'Fall', 'Winter', 'Spring' ]
>>> print(seasons.insert( 3 , 'Summer' ))
>>> print(seasons)
[ 'Fall', 'Winter', 'Spring', 'Summer' ]
```

• Remove element from the list

```python
>>> seasons.pop(2)
'Spring'
>>> print(seasons)
[ 'Fall', 'Winter', 'Summer' ]
```

The function `insert ( i , x)`, will insert the value `x` in the list at index `i`. If the list length < `i`, then the element will be added to the end of the list.

The function `pop ( i )`, removes the element at index `i` and returns its value.
Lists (Cont.)

• Finding index of an element in the list

```python
>>> seasons = ['Fall', 'Winter', 'Summer']
>>> seasons.index('Winter')
1
>>> seasons.index('Spring')
Error
```

The function `index(x)`, will return the index of the first occurrence of the element `x`.
An error is returned if the element is not in the list.

• Reverse elements in the list

```python
>>> seasons.reverse()
[ 'Summer', 'Winter', 'Fall' ]
```

The function `reverse()`, reverses the order of the elements in the list.

• Usage of list functions:

  [https://docs.python.org/2/tutorial/datastructures.html](https://docs.python.org/2/tutorial/datastructures.html)
Using **in** operator with lists

```python
# Lists of months with 31, 30, and 28 days
month31 = [ "JAN", "MAR", "MAY", "JULY", "AUG", "OCT", "DEC", "JANUARY", "MARCH" ]
month30 = [ "APR", "JUNE", "SEPT", "NOV" ]
month28 = [ "FEB" ]

# Print months
print ("Months: ", month31+month30+month28)

# User input
userInput = input("Enter month: ").upper()

# Check if input is in month31
if userInput in month31 :
    print (userInput, " has 31 days.")

# Check if input is in month30
elif userInput in month30 :
    print (userInput, " has 30 days.")

# Check if input is in month28
elif userInput in month28 :
    print (userInput, " has 28 days.")

# If not in any list
else :
    print ("You entered an invalid month. ")
```

Let's make a pizza

• Display all available toppings to the user, and then continually ask them to enter the toppings they want.

• The cost of a cheese pizza is $10.99. Each additional topping is $1.00.

• Compute the total cost of the pizza, and display it to the user.
Using **for-loop** to traverse a list

```python
months = [ "JAN", "FEB", "MAR", "APR", "MAY", "JUNE", "JULY", "AUG", "SEPT", "OCT", "NOV", "DEC" ]

for i in range(0, len(months), 2):
    print(months[i])
```

```python
months = [ "JAN", "FEB", "MAR", "APR", "MAY", "JUNE", "JULY", "AUG", "SEPT", "OCT", "NOV", "DEC" ]

for m in months:
    print(m)
```
Example

- Let's write a function called `capitalizeAll(t)` that receives a list of strings, and returns a new list with each string capitalized.

```python
def capitalizeAll ( list_of_strings ) :
    resultList = []
    for s in list_of_strings :
        resultList.append ( s.upper() )
    return resultList
```
Using **sum** with lists

grades = [100, 75, 80.5, 60, 90, 100, 40]

print ("The average is", sum(grades) / len(grades))

# lets drop the lowest grade
lowestGrade = min(grades)

print ("The lowest grade is : ", lowestGrade)

grades.remove(lowestGrade)
Revising Fibonacci to use lists

n = 10
fibSeq = [0, 1]

for i in range (2, n+1) :
    nextNum = sum( fibSeq[ i-2: ] )
    fibSeq.append( nextNum )
print (nextNum)
Remember nested loops??

- A loop inside another loop….thats all.

```python
for i in range(1, 3):
    for j in range(1, 3):
        print(i * j)
```

What does this code print?
What are nested loops??

• A loop inside another loop...that's all.

```python
for i in range(1, 3):
    for j in range(i, 3):
        print(i * j)
```

What does this code print?

Note, this is variable i, not the value 1
Nested Loop Example

• Add up all items in a loop.

```python
list1 = [ [ 1, 2, 3] , [4, 5, 6] ]
total = 0
for i in range (0, len(list1)) :
    for j in range ( 0, len (list1[0])) :
        total += list1[i][j]
```
Lists and Strings

- List and Strings are similar… but not the same thing
  - List is a sequence of values (any data-type)
  - String is a sequence of characters

```
s = "spam"
t = list(s)
print(t)
[ 's', 'p', 'a', 'm' ]
```

```
s = "spam spam spam"
t = s.split()
print(t)
[ 'spam', 'spam', 'spam' ]
```

list(s) splits the characters in a string and creates a list

s.split() splits the words in a string (using whitespace as a delimiter), and creates a list
Lists and Strings (Cont.)

```python
s = "spam-spam-spam"
t = s.split()
print(t)
['spam-spam-spam']

t = s.split('-')
print(t)
['spam', 'spam', 'spam']
```

`s.split(<delimiter>)` splits the words in a string (using the given delimiter), and creates a new list.
Aliasing when using lists

```python
listA = [1, 2, 3, 4, 5]
listB = listA

# if I change listA, the change will apply to listB also
# since they refer to the same object
listA[0] = 100
print(listB)
[100, 2, 3, 4, 5]
```