Lists, Strings, Tuples

NMT CSE/IT 107

Quiz

- Redo to boost grade. Pick up from office during office hours.
- Focus on Word Problems. Execute code you wrote as is in python.
- Flow charts. use boolean expressions
- Question 19. Valid commands are "forward", "backward", "left", "right"

Quiz

- Redo to boost grade. Pick up from office during office hours.
- Focus on Word Problems. Execute code you wrote as is in python.
- Flow charts. use boolean expressions
- Question 19. Valid commands are "forward", "backward", "left", "right"

Quiz

- Redo to boost grade. Pick up from office during office hours.
- Focus on Word Problems. Execute code you wrote as is in python.
- Flow charts. use boolean expressions
- Question 19. Valid commands are "forward", "backward", "left", "right"

Quiz

- Redo to boost grade. Pick up from office during office hours.
- Focus on Word Problems. Execute code you wrote as is in python.
- Flow charts. use boolean expressions
- Question 19. Valid commands are "forward", "backward", "left", "right"

Do's/Don'ts

Don't do this!!!

```
1  x = True
2  while x == True:
3      ...
4
5  if x == True:
6      ...
```

Do this!!!

```
1  x = True
2  while x:
3      ...
4
5  if x:
6      ...
```

Do's/Don'ts

Don't do this!!!

```
1  x = True
2  while x == True:
3      ...
4
5  if x == True:
6      ...
```

Do this!!!

```
1  x = True
2  while x:
3      ...
4
5  if x:
6      ...
```

Program Flow

- Difference between **defining** a function and **calling** a function
- Define —>def function_name():**The body of the function is not executed**
- Call —>function_name() **Execute function_name, i.e. run the body of the function**

demo my_program.py

Program Flow

- Difference between **defining** a function and **calling** a function
- Define —>def function_name():**The body of the function is not executed**
- Call —>function_name() **Execute function_name, i.e. run the body of the function**

demo my_program.py

Program Flow

- Difference between **defining** a function and **calling** a function
- Define —>def function_name():**The body of the function is not executed**
- Call —>function_name() **Execute function_name, i.e. run the body of the function**

demo my_program.py

Program Flow

- Difference between **defining** a function and **calling** a function
- Define —>def function_name():**The body of the function is not executed**
- Call —>function_name() **Execute function_name, i.e. run the body of the function**

demo my_program.py

Lists

- Lists are mutable sequences of items
- listA = [1,2,3,4]
- empty_list = []
- Lists can contain any object and any mix of data types
- listA = [1, '2', 3, True]
- get the length of a list `len(listA)`
- Access individual items of the list. Lists are zero - indexed.
- first_item = listA[0] # set first_item to first element of list x
- last_item = listA[-1] # set last_item to last element of list x
- How would you retrieve the second to last element of a list

Lists

- Lists are mutable sequences of items
- listA = [1,2,3,4]
- empty_list = []
- Lists can contain any object and any mix of data types
- listA = [1, '2', 3, True]
- get the length of a list `len(listA)`
- Access individual items of the list. Lists are zero - indexed.
- first_item = listA[0] # set first_item to first element of list x
- last_item = listA[-1] # set last_item to last element of list x
- How would you retrieve the second to last element of a list

Lists

- Lists are mutable sequences of items
- listA = [1,2,3,4]
- empty_list = []
- Lists can contain any object and any mix of data types
- listA = [1, '2', 3, True]
- get the length of a list `len(listA)`
- Access individual items of the list. Lists are zero - indexed.
- first_item = listA[0] # set first_item to first element of list x
- last_item = listA[-1] # set last_item to last element of list x
- How would you retrieve the second to last element of a list

Lists

- Lists are mutable sequences of items
- listA = [1,2,3,4]
- empty_list = []
- Lists can contain any object and any mix of data types
- listA = [1, '2', 3, True]
- get the length of a list `len(listA)`
- Access individual items of the list. Lists are zero - indexed.
- first_item = listA[0] # set first_item to first element of list x
- last_item = listA[-1] # set last_item to last element of list x
- How would you retrieve the second to last element of a list

Lists

- Lists are mutable sequences of items
- listA = [1,2,3,4]
- empty_list = []
- Lists can contain any object and any mix of data types
- listA = [1, '2', 3, True]
- get the length of a list `len(listA)`
- Access individual items of the list. Lists are zero - indexed.
- first_item = listA[0] # set first_item to first element of list x
- last_item = listA[-1] # set last_item to last element of list x
- How would you retrieve the second to last element of a list

Lists

- Lists are mutable sequences of items
- listA = [1,2,3,4]
- empty_list = []
- Lists can contain any object and any mix of data types
- listA = [1, '2', 3, True]
- get the length of a list `len(listA)`
- Access individual items of the list. Lists are zero - indexed.
- first_item = listA[0] # set first_item to first element of list x
- last_item = listA[-1] # set last_item to last element of list x
- How would you retrieve the second to last element of a list

Lists

- Lists are mutable sequences of items
- listA = [1,2,3,4]
- empty_list = []
- Lists can contain any object and any mix of data types
- listA = [1, '2', 3, True]
- get the length of a list `len(listA)`
- Access individual items of the list. Lists are zero - indexed.
- first_item = listA[0] # set first_item to first element of list x
- last_item = listA[-1] # set last_item to last element of list x
- How would you retrieve the second to last element of a list

Lists

- Lists are mutable sequences of items
- listA = [1,2,3,4]
- empty_list = []
- Lists can contain any object and any mix of data types
- listA = [1, '2', 3, True]
- get the length of a list `len(listA)`
- Access individual items of the list. Lists are zero - indexed.
- first_item = listA[0] # set first_item to first element of list x
- last_item = listA[-1] # set last_item to last element of list x
- How would you retrieve the second to last element of a list

Lists

- Lists are mutable sequences of items
- listA = [1,2,3,4]
- empty_list = []
- Lists can contain any object and any mix of data types
- listA = [1, '2', 3, True]
- get the length of a list `len(listA)`
- Access individual items of the list. Lists are zero - indexed.
- first_item = listA[0] # set first_item to first element of list x
- last_item = listA[-1] # set last_item to last element of list x
- How would you retrieve the second to last element of a list

Lists

- Lists are mutable sequences of items
- listA = [1,2,3,4]
- empty_list = []
- Lists can contain any object and any mix of data types
- listA = [1, '2', 3, True]
- get the length of a list `len(listA)`
- Access individual items of the list. Lists are zero - indexed.
- first_item = listA[0] # set first_item to first element of list x
- last_item = listA[-1] # set last_item to last element of list x
- How would you retrieve the second to last element of a list

Lists: Add

- Lists are mutable.
- Change value of the first position (i.e. index=0).
  listA[0] = 134
- Add element to the end of a list.
  `listA.append(100)`
- Insert element.
  `listA.insert(0, 'Hello')`
- Concatenate Lists.
  `listA.extend([1,2,3,4])`

Lists: Add

- Lists are mutable.
- Change value of the first position (i.e. index=0).
  listA[0] = 134
- Add element to the end of a list.
  `listA.append(100)`
- Insert element.
  `listA.insert(0, 'Hello')`
- Concatenate Lists.
  `listA.extend([1,2,3,4])`

Lists: Add

- Lists are mutable.
- Change value of the first position (i.e. index=0).
  listA[0] = 134
- Add element to the end of a list.
  `listA.append(100)`
- Insert element.
  `listA.insert(0, 'Hello')`
- Concatenate Lists.
  `listA.extend([1,2,3,4])`

Lists: Add

- Lists are mutable.
- Change value of the first position (i.e. index=0).
  listA[0] = 134
- Add element to the end of a list.
  `listA.append(100)`
- Insert element.
  `listA.insert(0, 'Hello')`
- Concatenate Lists.
  `listA.extend([1,2,3,4])`

Lists: Add

- Lists are mutable.
- Change value of the first position (i.e. index=0).
  listA[0] = 134
- Add element to the end of a list.
  `listA.append(100)`
- Insert element.
  `listA.insert(0, 'Hello')`
- Concatenate Lists.
  `listA.extend([1,2,3,4])`

Lists: Remove

- Lists are mutable.
- Remove first occurrence of an item. listA.remove(1)
- Remove and return element at index. x = listA.pop(1)

demo in terminal

Lists: Remove

- Lists are mutable.
- Remove first occurrence of an item. listA.remove(1)
- Remove and return element at index. x = listA.pop(1)

demo in terminal

Lists: Remove

- Lists are mutable.
- Remove first occurrence of an item. listA.remove(1)
- Remove and return element at index. x = listA.pop(1)

demo in terminal

Tuples

- Tuples are similar to Lists but Tuples are immutable.

```
1 tupleA = (1,2,3)
2 tupleA = 1, 2, 3
3 x, y, z = 1,10,20
```

- Tuples are immutable. **Tuples cannot be changed**

Not valid

```
1 tupleA[0] = 10
```

- Strings are also immutable.

Not Valid

```
1 message = 'hello World'
2 message[0] = 'H'
```

Tuples

- Tuples are similar to Lists but Tuples are immutable.

```
1 tupleA = (1,2,3)
2 tupleA = 1, 2, 3
3 x, y, z = 1,10,20
```

- Tuples are immutable. **Tuples cannot be changed**

Not valid

```
1 tupleA[0] = 10
```

- Strings are also immutable.

Not Valid

```
1 message = 'hello World'
2 message[0] = 'H'
```

Tuples

- Tuples are similar to Lists but Tuples are immutable.

```
1 tupleA = (1,2,3)
2 tupleA = 1, 2, 3
3 x, y, z = 1,10,20
```

- Tuples are immutable. **Tuples cannot be changed**

Not valid

```
1 tupleA[0] = 10
```

- Strings are also immutable.

Not Valid

```
1 message = 'hello World'
2 message[0] = 'H'
```

Tuples

- Tuples are similar to Lists but Tuples are immutable.

```
1 tupleA = (1,2,3)
2 tupleA = 1, 2, 3
3 x, y, z = 1,10,20
```

- Tuples are immutable. **Tuples cannot be changed**

Not valid

```
1 tupleA[0] = 10
```

- Strings are also immutable.

Not Valid

```
1 message = 'hello World'
2 message[0] = 'H'
```

Tuples

- Tuples are similar to Lists but Tuples are immutable.

```
1 tupleA = (1,2,3)
2 tupleA = 1, 2, 3
3 x, y, z = 1,10,20
```

- Tuples are immutable. **Tuples cannot be changed**

Not valid

```
1 tupleA[0] = 10
```

- Strings are also immutable.

Not Valid

```
1 message = 'hello World'
2 message[0] = 'H'
```

Tuples

- Tuples are similar to Lists but Tuples are immutable.

```
1 tupleA = (1,2,3)
2 tupleA = 1, 2, 3
3 x, y, z = 1,10,20
```

- Tuples are immutable. **Tuples cannot be changed**

Not valid

```
1 tupleA[0] = 10
```

- Strings are also immutable.

Not Valid

```
1 message = 'hello World'
2 message[0] = 'H'
```

for loops

- for loops move through a collection (list, tuple, string) one element at at time.
- for *element* in *collection*:
    body

```
1 for letter in "Hello World":
2     print(letter)
3
4 words = "Hello World".split(' ')
5 # words = ['Hello', 'World']
6 for word in words:
7     print(word)
```

for loops

- for loops move through a collection (list, tuple, string) one element at at time.
- for *element* in *collection*:
  body

```
for letter in "Hello World":
    print(letter)

words = "Hello World".split(' ')
# words = ['Hello', 'World']
for word in words:
    print(word)
```

for loops

- for loops move through a collection (list, tuple, string) one element at at time.
- for *element* in *collection*:
  body

```
1  for letter in "Hello World":
2      print(letter)
3
4  words = "Hello World".split(' ')
5  # words = ['Hello', 'World']
6  for word in words:
7      print(word)
```

Exercise: Search

Given a sentence find the first word that starts with the letter 'f'.

Exercise: Search

Draw a flow diagram.

```
1 sentence = 'The quick furry Fox jumps over the lazy dog'
2 words = sentence.split(' ') # split sentence into words
3 first_f_word = None
4 for word in words: # iterate words
5     if word[0] == 'f': # test condition
6         first_f_word = word
7         break
8 print('The first "f" word is {}'.format(first_f_word))
```

Exercise: Search

Draw a flow diagram.

```
1  sentence = 'The quick furry Fox jumps over the lazy dog'
2  words = sentence.split(' ') # split sentence into words
3  first_f_word = None
4  for word in words: # iterate words
5      if word[0] == 'f': # test condition
6          first_f_word = word
7          break
8  print('The first "f" word is {}'.format(first_f_word))
```

Tuple vs List

- When should you use a tuple and when a list
- Use tuple if you want a constant sequence of items.
- `breakfast_menu_items = ('spam', 'eggs')`
- Use list when you want a sequence of items that can grow and shrink and change

Tuple vs List

- When should you use a tuple and when a list
- Use tuple if you want a constant sequence of items.
- `breakfast_menu_items = ('spam', 'eggs')`
- Use list when you want a sequence of items that can grow and shrink and change

Tuple vs List

- When should you use a tuple and when a list
- Use tuple if you want a constant sequence of items.
- `breakfast_menu_items = ('spam', 'eggs')`
- Use list when you want a sequence of items that can grow and shrink and change

Tuple vs List

- When should you use a tuple and when a list
- Use tuple if you want a constant sequence of items.
- `breakfast_menu_items = ('spam', 'eggs')`
- Use list when you want a sequence of items that can grow and shrink and change

Tuple vs List

Find all words that start with 'a'

```
a_words = []
for word in ('abc', 'bar', 'foo', 'a bat'):
    if word[0] == 'a':  # word.startswith('a')
        a_words.append(word)

num_a = len(a_words)

print('Number of words '\
      'starting with a = {}'.format(num_a))
print(a_words)
```

Tuple vs List

Find all words that start with 'a'

```
1  a_words = []
2  for word in ('abc', 'bar', 'foo', 'a bat'):
3      if word[0] == 'a':  # word.startswith('a')
4          a_words.append(word)
5
6  num_a = len(a_words)
7
8  print('Number of words '\
9            'starting with a = {}'.format(num_a))
10 print(a_words)
```

Boolean Expressions

- Boolean expressions evaluate to `True` or `False`
- Simple boolean expressions use relational operators for the test: `<`, `<=`, `>`, `>=`, `==`, `!=`
- There are two more boolean operators. **in**, and **is**.

Boolean Expressions

- Boolean expressions evaluate to `True` or `False`
- Simple boolean expressions use relational operators for the test: `<`, `<=`, `>`, `>=`, `==`, `!=`
- There are two more boolean operators. **in**, and **is**.

Boolean Expressions

- Boolean expressions evaluate to `True` or `False`
- Simple boolean expressions use relational operators for the test: $<$, $<=$, $>$, $>=$, `==`, `!=`
- There are two more boolean operators. **in**, and **is**.

in keyword

- The `in` keyword is used to check whether a value is contained inside another object such as a string or list.
- `in` returns True or False

in keyword

- The `in` keyword is used to check whether a value is contained inside another object such as a string or list.
- `in` returns True or False

Examples: In

```
1 colors = ['red','yellow','green']
2 if 'red' in colors:
3     print('Red is in colors')
4 else:
5     print('Red is not in colors')
```

```
1 valid_commands = ['forward', 'backward', 'left', 'right']
2 command = input('Please enter a command ')
3 if command in valid_commands:
4     print('Command is valid')
5 else:
6     print('Command is not valid')
```

Examples: In

```
1 colors = ['red','yellow','green']
2 if 'red' in colors:
3    print('Red is in colors')
4 else:
5    print('Red is not in colors')
```

```
1 valid_commands = ['forward', 'backward', 'left', 'right']
2 command = input('Please enter a command ')
3 if command in valid_commands:
4     print('Command is valid')
5 else:
6     print('Command is not valid')
```