Computer Code for Beginners
Week 3

Matt Luckcuck

5th of October 2017

Last Time

Previously...

- Loops
- Lists
- Functions

Outline

Outline

- More Sequences
  - List Recap
  - Strings
  - Tuples
- More Functions

More Sequences

Sequences

List Recap

- A sequence is an ordered set of data
- Last week: lists - a basic sequence type
- `colours = ["Red", "Blue", "Green"]`
- Lists can be changed...
  - We say they are *mutable*
  - `colours[0] = "Purple"`
  - `colours.append("Orange")`

Sequences

List Recap

- A sequence is an ordered set of data
- Last week: lists - a basic sequence type
- `colours = ["Red", "Blue", "Green"]`
- Lists can be changed...
  - We say they are *mutable*
  - `colours[0] = "Purple"`
  - `colours.append("Orange")`
  - `colours = ["Purple", "Blue", "Green", "Orange"]`

Sequences

Nested Lists

- We can have nested lists
- Each item of the list... is a list
  - `gameBoard = [ [1,2,3], [1,2,3], [1,2,3] ]`
- This is a 2-dimensional List
  - We could have 3... n dimensional Lists
- To loop through every element in an n-dimensional list, we need n loops
  - For a 2-d list like `gameBoard` we need two loops

Sequences

List

- Support some operations common to sequences
  - `colours = ["Purple", "Blue", "Green", "Orange"]`
- Indexing – `colours[0]`
- Length – `len(colours)`
- Slicing – `s[0:2]`

Sequences

List

- Support some operations common to sequences
  - `colours = ["Purple", "Blue", "Green", "Orange"]`
- Indexing – `colours[0]`
- Length – `len(colours)`
- Slicing – `s[0:2]`
  - `["Purple", "Blue"]`

Sequences

String

- Another sequence type
  - Each character is zero-indexed by a number
- `s = "Purple"`
- Allows some sequence operations:
  - Indexing – `s[0]`
  - Slicing – `s[0:2]`
- But Strings are *immutable*

Sequences

String

- Another sequence type
  - Each character is zero-indexed by a number
- `s = "Purple"`
- Allows some sequence operations:
  - Indexing – `s[0]`
    - `"P"`
  - Slicing – `s[0:2]`
- But Strings are *immutable*

Sequences

String

- Another sequence type
  - Each character is zero-indexed by a number
- `s = "Purple"`
- Allows some sequence operations:
  - Indexing – `s[0]`
    - `"P"`
  - Slicing – `s[0:2]`
    - `"Pu"`
- But Strings are *immutable*

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?
  - 170

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?
  - 170
- `t[1]` is?

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?
  - 170
- `t[1]` is?
  - 52

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?
  - 170
- `t[1]` is?
  - 52
- What about `t[0] = 7`?

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?
  - 170
- `t[1]` is?
  - 52
- What about `t[0] = 7`?
  - `TypeError: "tuple"object does not support item assignment`

Sequences

Tuples

- Useful for passing around related data
  - Height and Weight (Pair)
  - X, Y, and Z coordinates (Triple)
  - Car tyre pressures (4 Tuple)
- Again, using them is a design decision

More Functions

Last Time...

Function

- Block of code wrapped up that does something for us
  - Function defined with: `def funcName():`
  - Function called using `funcName()`
- Function body is an indented block
- We've seen some built-in functions:
  - `print()`
  - `len()`
  - `range()`

Last Time...

```
1 def add(a, b):
2   """ Adds a to b """
3   return a+b
4
5 result = add(2,2)
```

Last Time...

Function

- We can pass data into a function...
  - Called *parameters*
- Functions can *read* variables defined outside
  - More one this next week...
- Function may pass us back some data...
  - Called the *return value*
  - Imagine the return value replacing the call to the function
- Functions with no `return` statement return `None`
  - `None` is a type that represents nothing

Functions

Functions and Variable Scope

- Variables defined outside of a function are *global*
- Variables defined inside a function are *local* to the function
  - Use `return` to pass variables out of a function
- This is known as a variable's *scope*

```
1 def func1():
2   var1 = 10
3 print(var1) # this wont work
```

Functions

Functions and Variable Scope

- Variables defined outside of a function are *global*
- Variables defined inside a function are *local* to the function
  - Use `return` to pass variables out of a function
- This is known as a variable's *scope*

```
1 def func1():
2   var1 = 10
3 print(var1) # this wont work
```

- `NameError: name 'var1'is not defined`

```
1 var2 = 10
2
3 def func2():
4   print(var2) # this is fine
```

Functions

Functions and Scope

- A local variable can share the name of a global variable
  - But the global variable keeps its value

```
1 var3 = 10
2
3 def func3():
4   var3 = 20
5   print(var3) # prints 20
6
7 func3()
8 print(var3) # prints 10
```

Functions

Functions and Scope

- Global variables can be used inside a function
  - But, needs `global` keyword if we want to alter it

```
1 var4 = 10
2
3 def func4():
4
5   var4 = var4 + 10 # Wont work
```

Functions

Functions and Scope

- Global variables can be used inside a function
  - But, needs `global` keyword if we want to alter it

```
1 var4 = 10
2
3 def func4():
4
5   var4 = var4 + 10 # Wont work
```

- `UnboundLocalError:local variable 'var4'referenced before assignment`

Functions

Functions and Scope

- Global variables can be used inside a function
  - But, needs `global` keyword if we want to alter it

```
1 var4 = 10
2
3 def func4():
4   global var4
5   var4 = var4 + 10 # Now it's fine
```

Functions

Return Values

- Usually one value...
  - E.g. `return x`
  - Called *single return*

```
1 var5 = 10
2
3 def func5(var):
4   return var + 10
5
6 var5 = func5(var5)
7 print(var5) # prints 20
```

Functions

Return Values

- But Python allows *multiple returns*
  - `return x, y`
  - `a, b = multiReturnFunc()`
- *Technically* this is returning a tuple
- This can be useful, but...
  - Be careful it doesn't get too confusing
  - Not always available in other languages, so don't rely on it
  - What data type could we use to return multiple values?

Functions

Return Values

- But Python allows *multiple returns*
  - `return x, y`
  - `a, b = multiReturnFunc()`
- *Technically* this is returning a tuple
- This can be useful, but...
  - Be careful it doesn't get too confusing
  - Not always available in other languages, so don't rely on it
  - What data type could we use to return multiple values?
    - List

Functions

Return Values

- But Python allows *multiple returns*
  - `return x, y`
  - `a, b = multiReturnFunc()`
- *Technically* this is returning a tuple
- This can be useful, but...
  - Be careful it doesn't get too confusing
  - Not always available in other languages, so don't rely on it
  - What data type could we use to return multiple values?
    - List
    - Tuple(s) Explicitly

Functions

Return Values

- But Python allows *multiple returns*
  - `return x, y`
  - `a, b = multiReturnFunc()`
- *Technically* this is returning a tuple
- This can be useful, but...
  - Be careful it doesn't get too confusing
  - Not always available in other languages, so don't rely on it
  - What data type could we use to return multiple values?
    - List
    - Tuple(s) Explicitly
    - There are others

Functions

Manipulating Parameters

- Function parameters are ‘passed by value’
  - Value of a number is the number
  - Value of a list is a *reference* to the list
- What does this mean for us?

Functions

Passing a number

- Passing a number...
- Function updates number...
- Original is unaltered

```
1 var5 = 10
2
3 def func5(var):
4   return var + 10
5
6 func5(var5)
7 print(var5) # prints 10
```

Functions

Passing a list

- Passing a list...
- Function updates the list
- The global copy will be updated

```
1 var6 = [10,20,30]
2
3 def func6(theList):
4   theList.append(40)
5
6 func6(var6)
7 print(var6) # prints [10, 20, 30, 40]
```

Functions

Final Word...

- While you're getting your head around this...
- Stick to *reading* variables defined outside your function
- Passing *parameters* into your function, and
- *Returning* values from your function
- Avoid writing to global variables

Summary

Summary

Summary

- Sequences are **ordered** collections of items
- List
- Tuple
  - New sequence type
- String
- In-Depth of how Functions Work

Exercises

- `global` Variable Swap
- Mine Detector