Data types and Conditionals

NSC 1002 Workshop 2

04/11/2020

Geoff Vallis

Outline

Python Program

1. Bring up the Spyder editor
2. At the top of the program, using the docstring:

   ```
   """
       Description here
   """
   ```

   describe what the program does.
3. Advisory: sketch how you want your program to work on a piece of paper.
4. Type the commands into the Spyder editor.
5. Run the program using the tabs at the top of the Spyder window, or use F5.

Lists

1. ‘Lists’ are a ‘mutable ordered sequence’. Mutable means we can change the elements in them!

2. They are defined by enclosure in square brackets [ ]

   ```
   list0=[30.0, 30.5, 33.0, 'dog']
   print(list0[2])
   33.0
   list0[2] = 31.0
   print(list0)
   [30.0, 30.5, 31.0, 'dog']
   ```

3. List elements are **indexed**, starting from 0

4. So the *first* element in our list `list0[0]`, and the *last* element is `list0[3]`

5. Python also allows us to count backwards from the end of a list using negative indices: `list0[-1]`

Why are lists useful? Example, make a plot!

```
"""
An example plotting program.
Create two lists, plot them.
@author: mk450
"""
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [1, 7, 3, 5]
plt.plot(x, y)
plt.show()
```

Let's experiment by changing one of the values in each list!

Tuples

- Tuples an immutable ordered sequence. Once defined, the elements in a list are fixed.
- They are enclosed in round brackets ( )

```
t0=(1, 'cat', 32.1)
print(t0[2])
32.1
t0[1] = 'cat'
```

Tuples

- Tuples an immutable ordered sequence. Once defined, the elements in a list are fixed.
- They are enclosed in round brackets ( )

```
t0=(1, 'cat', 32.1)
print(t0[2])
32.1
t0[1] = 'cat'
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

- To access a particular element you always use a square bracket, like t0[2].
- The first element is t0[0] not t0[1]! (A peculiarity of Python (and C)!)

Strings

- Strings are ordered sequences of elements.
- They can be represented by single or double quotes.
    - But you must be consistent (e.g. `s = ‘Hello’’` is not okay, while `s = ‘Hello’` and `s = ‘‘Hello’’` are fine.)
- Strings are **immutable**, which means individual elements cannot be changed once set

Strings

```
s = 'Hello'
print(s[0])
Output is H
s[0] = 'h'
```

Will this result in an error?

Strings

```
s = 'Hello'
print(s[0])
Output is H
s[0] = 'h'
```

Will this result in an error?

What about

`s='hello'`

Strings

- Operations can be performed on strings e.g.:

  ```
  s1 = 'Hello'
  s2 = 'world'
  s3 = s1 + s2
  s4 = 2 * s1 + s2

  s1.isupper()
  s2.islower()
  s1.swapcase()
  ```

- To find out more about these options `dir(s1)`, will return a *list* of them.

Conditional statements

- Conditionals are how to branch your code, they contain three parts:
  - A test that will result in a True or False answer
  - Code that executes when this answer is True
  - Code that executes when this answer is False (*optional*)

```
if <conditional statement>:
    Do Something
elif <conditional statement>:
    Do some other thing
else:
    Do something different still
```

Operators

- Relational operators used in conditional statements: $== \quad > \quad < \quad != \quad >= \quad <=$.
- Logicals used in conditional statements: *and*, *or*, and *not*
- Example: `a != b or c == b`

Operators

- Relational operators used in conditional statements:
  $== \quad > \quad < \quad != \quad >= \quad <=.$
- Logicals used in conditional statements: *and*, *or*, and *not*
- Example: `a != b or c == b`

```
if a != b or c == b:
        Do Something
pass
```

Will be true when either *a* does not equal *b*, OR *b* is equal to *c*

Note: the command 'pass' does nothing. It is useful sometimes, for example to indicate the end of an indented block.

Examples

```
if x = = y:
    print('x equals y')
else:
    if x > y:
        print('x is greater than y')
    else:
        print('x less than y')
```

Questions?

Questions?

Now work through the examples