

TYPES, LISTS, AND STRINGS

CSSE 120 – Rose-Hulman Institute of Technology

Outline



- More on Numeric Types
- Long Integers vs. Floats
- Type Conversion
- Lists and Strings
- Lab Time

Program Grade Components

Percent	Feature
≥ 70	<p>Correctness:</p> <p>The program accomplishes what the assignment specifies</p>
≤ 15	<p>Documentation:</p> <p>Comments at beginning of the program. Your name, what the program does</p> <p>How the program is to be run (interactive or reads a file; if the latter, what is its format?)</p> <p>Doc comments for classes and functions</p> <p>Internal comments for any parts of the program that may not be obvious to a human reader</p>
≤ 15	<p>Style/maintainability:</p> <p>Sensible variable and function names</p> <p>No magic numbers</p> <p>Reasonable decomposition into functions, classes, methods</p> <p>Sensible SVN commit messages</p>

Seeing Your Grades in ANGEL



- ❑ In the CSSE 120 ANGEL course, choose the REPORTS tab
- ❑ Under CATEGORY, choose GRADES
- ❑ Click RUN
- ❑ You will have to scroll down to see some of your grades

Numeric Types - Recap

- int : integer type
 - ▣ Exact values – limited range
 - ▣ An operation on two ints **always** yields an int
- float : real number type
 - ▣ Approximate values – much larger range
 - ▣ An operation on float and int yields a float

```
>>> 5/3
1
>>> 5.0/3
1.6666666666666667
>>> 5/2
2
>>> 5/2.0
2.5
>>> 5%3
2
>>> 5%2
1
>>> 5.0//2.0
2.0
```

Integer Representations

- An **int** is represented by a fixed-length sequence of bits
 - A **bit** is a **binary digit**: its value is either 0 or 1.
- On typical 2009 architectures, that length is 32
- How many different values can be represented by **n** bits?
- Thus there is a largest **int** value
- How to deal with larger integer values?
 - Use floats? What could be wrong with that?
 - Do what other languages do? (overflow)

Python's **long** integer type

- Allows arbitrarily large integers
- Automatically created when needed:

```
>>> 10**10  
10000000000L
```
- You can specify a long literal

```
>>> 4L/2  
2L  
>>> type(4L)  
<type 'long'>
```
- Since **long** covers all integers (up to the memory limits of the computer) why have an **int** type at all?
 - Why not use **long** for all integer calculations?

Type Conversions

- Sometimes we have a value of one type, but we need the corresponding value of another type
- In some cases, conversion is automatic:
 - $x = 3$
 $y = x/7.5$
- Python provides functions that allow you to explicitly convert data to another type
 - `int()`
 - `float()`
 - `str()`

Hidden example slide

```
>>> x = 7
>>> x/2
>>> float(x)/2
>>> sum = 0
>>> for i in range (1,12):
    sum = sum + 1/i
>>> sum
>>> sum = 0
>>> for i in range (1,12):
    sum = sum + 1/float(i)
>>> sum
>>> int(3.5)
>>> int(-3.7)
>>> int(17.0 / 6.0)
>>> float(17 / 6)
>>> str(6.7)
>>>int('6.4')
>>> float(int(3.5))
>>> float("34" + "4.2")
```

Also,
str(8)
int('8')

In-class Exercise



- Please download from ANGEL:
 - Lessons > Modules to Download in Class > Session 4 > session04.py
 - Do the practiceNumberTypes section.

Sequences in Python

- A sequence is an ordered collection of data items.

There are two kinds:

- List: mutable [3, 4, 6]

- Tuple: immutable (3, 4, 6)

- Simple examples of generating lists and tuples:

- `>>> range(4, 11, 2)`
[4, 6, 8, 10]

- `>>> 3*4, 3-4, 3+4, 3/4`
(12, -1, 7, 0)

Slices of a List

- **`list[m:n]`** returns a new list consisting of `[list[m], list[m+1], list[m+2], ... list[n-1]]`
- **`list[:n]`** returns a new list consisting of `[list[0], list[1], ... list[n-1]]`
- **`list[m:]`** returns a new list consisting of all elements of `list` beginning with `list[m]`.
- **`list[m:n:k]`**, similar to `range(m, n, k)`, returns a new list consisting of **every k^{th} element** of `list`, starting with `list[m]`.

Sequence Operations

- **len(<sequence>)**
 - ▣ Returns length of the sequence
- **<sequence>.index(<expr>)**
 - ▣ Returns the index of the first occurrence of the expression in the sequence
- **+** does concatenation
 - ▣ $[1, 2] + [7, 5]$ is $[1, 2, 7, 5]$
 - ▣ $(4,1) + (65, 2)$ is $(4,1, 65, 2)$

List-specific Operations

- `<list>.append (<expr>)`
 - ▣ Modifies the list by adding the value of the expression to the end of the list
- `<list>.reverse()`
 - ▣ Modifies the list by reversing the order of its elements
- `<list>.sort()`
 - ▣ Modifies the list by sorting the elements into increasing order
- Can you see why these operations don't work with tuples?
- Please do practice `WithLists` from `session04.py`

Not all expressions return values

- `>>> numList = [2, 5, 7, 2, 8, 4, 2, 6]`
- `>>> c = numList.count(2)`
`>>> c`
3
- `>>> r = numList.reverse()`
`>>> numList`
[6, 2, 4, 8, 2, 7, 5, 2]
- `>>> r`
- `>>> [r]`
[None]

Strings (character strings)

- **String literals:**

- `"One\nTwo\nThree"`

- `"Can't Buy Me Love"`

- `'I say, "Yes." You say, "No." '`

- `"'A double quote looks like this \", ' he said."`

- `""""I don't know why you say "Goodbye," I say "Hello." """`

String Operations

- Many of the operations listed in the book, while they work in Python 2.5, have been superseded by newer ones
- + is used for **String concatenation**: "xyz" + "abc"
- * is used for **String duplication**: "xyz " * 4
 - ```
>>> franklinQuote = 'Who is rich? He who is content. ' +
'Who is content? Nobody.'
```
  - ```
>>> franklinQuote.lower()  
'who is rich? he who is content.  who is content?  nobody.'
```
 - ```
>>> franklinQuote.replace('He', 'She')
'Who is rich? She who is content. Who is content? Nobody.'
>>> franklinQuote.find('rich')
```
  - 7

# Strings as Sequences

- A string is an **immutable** sequence of characters
- `>>> alpha = "abcdefg "`
- `>>> alpha[2]`
- `>>> alpha[1:4]`
- `>>> alpha[3] = "X" # illegal!`

# Strings and Lists

- A String method: **split** breaks up a string into separate words

- `>>> franklinQuote = 'Who is rich? He who is content. ' + 'Who is content? Nobody.'`

- `>>> myList = franklinQuote.split()  
['Who', 'is', 'rich?', 'He', 'who', 'is', 'content.',  
'Who', 'is', 'content?', 'Nobody.']`

- A string method: **join** recreates a list

- `'#'.join(myList)`

- `'Who#is#rich?#He#who#is#content.#Who#is#content?#Nobody.'`

- What is the value of `myList[0][2]`?

- Finish **practiceWithStringsAndLists**

# Lists of Strings

```
>>> beatles = ['John', 'Paul']
>>> beatles.append('George')
>>> beatles
['John', 'Paul', 'George']
>>> beatles + ['Ringo']
['John', 'Paul', 'George', 'Ringo']
>>> beatles
['John', 'Paul', 'George']
>>> beatles = beatles + ['Ringo']
>>> beatles
['John', 'Paul', 'George', 'Ringo']
>>> beatles[1]
'Paul'
>>> beatles[1][2]
'u'
```

# Optional: A Loop to Make a List

- Python's fancy term for this: **list comprehension**
- ```
>>> [i*i for i in range(6)]  
[0, 1, 4, 9, 16, 25]
```
- ```
>>> [[i, i*i] for i in range(5)]
[[0, 0], [1, 1], [2, 4], [3, 9], [4, 16]]
```
- Can you write a list comprehension for the value of cosine every 45 degrees around a circle?

# A List of Points

```
from zellegraphics import *
```

```
win = GraphWin()
```

```
pointList = [Point(30, 120), Point(150, 55), Point(80, 175)]
```

```
poly = Polygon(pointList)
```

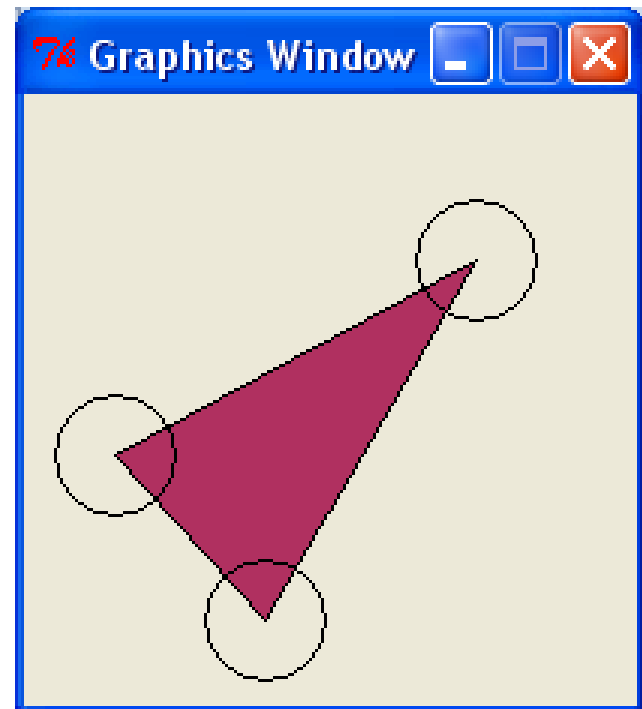
```
poly.setFill('maroon')
```

```
poly.draw(win)
```

```
for point in pointList:
```

```
 circ = Circle(point, 20)
```

```
 circ.draw(win)
```



# Homework 4



- See instructions linked from Course Schedule
- Upload solutions to dropboxes on ANGEL
- Once you "get the hang" of problems 3 and 4, you should probably start on *Pizza*, *Polygon*, and *Star* while we're here to help
- Make sure you configure Eclipse and PyDev for next class, if you desire some bonus points
  - details in HW4 instructions