As you arrive:

1. Start up your computer and plug it in
2. ***Log into Angel*** and go to CSSE 120
3. Do the ***Attendance Widget*** – the PIN is on the board
4. Go to the course ***Schedule Page***
   - From your *bookmark*, or from the *Lessons* tab in Angel
5. Open the ***Slides*** for today if you wish

Software Dev. Exercise

Day of year module

Character Strings

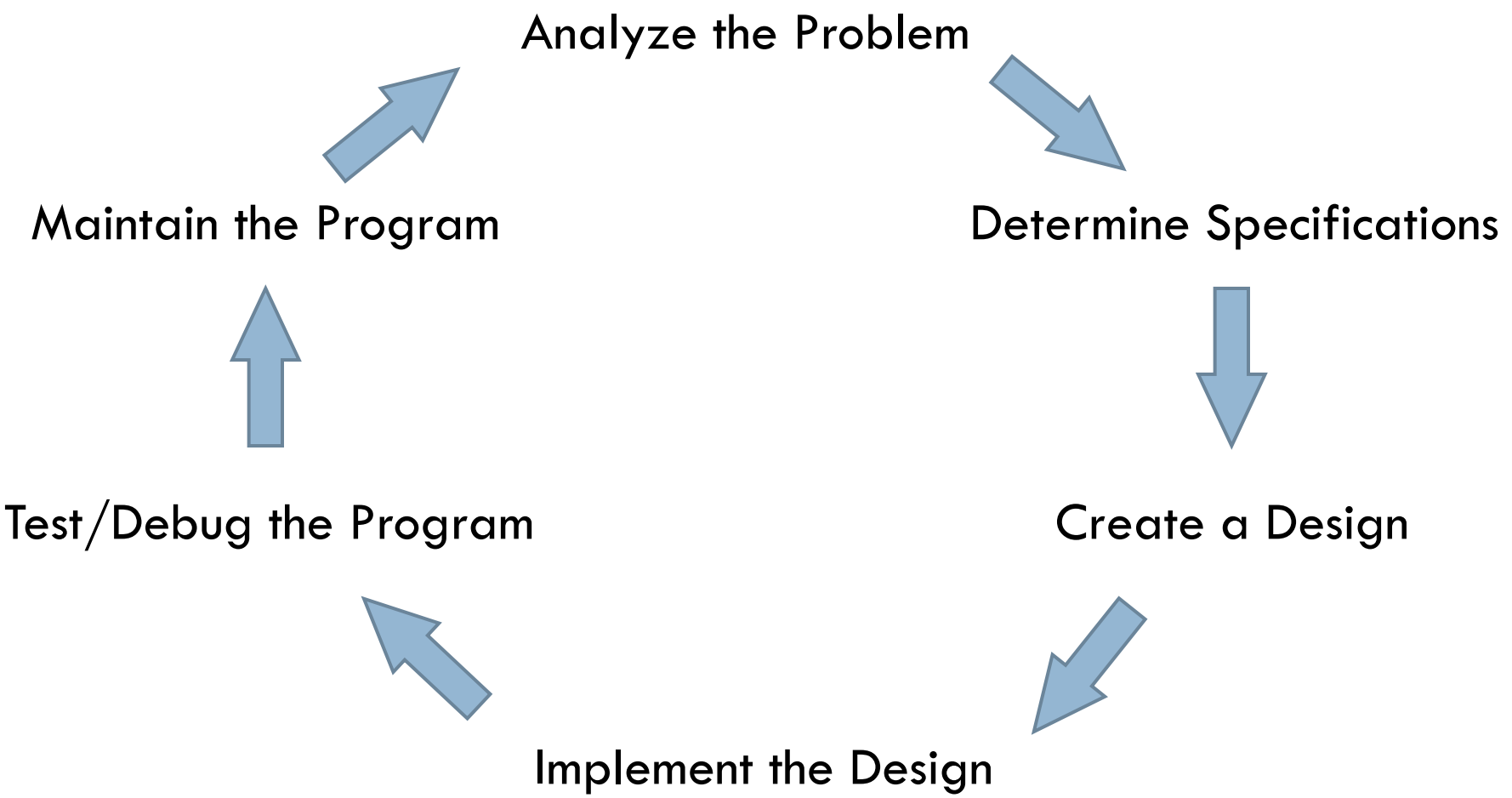
String operations
Lists and strings
String encodings
String formatting

Day, Month → Day of year

- When calculating the amount of money required to pay off a loan, banks often need to know what the "ordinal value" of a particular date is
  - For example, **March 6** is the **65th** day of the year (in a non-leap year)
- We need a program to calculate the day of the year when given a particular month and day

Q1

The Software Development Process



Phases of Software Development

- **Analyze:** figure out exactly what the problem to be solved is
- **Specify:** WHAT will program do? NOT HOW.
- **Design:** SKETCH how your program will do its work, design the algorithm
- **Implement**: translate design to computer language
- **Test/debug**: See if it works as expected.
  bug == error, debug == find and fix errors
- **Maintain**: continue developing in response to needs of users

Checkout today's project

- Go to SVN Repository view, at bottom of the workbench
  - If it is not there,
    Window→Show View→Other→SVN → SVN Repositories
- Browse SVN Repository view for
    06-StringsAndLists project
- Right-click it, and choose Checkout
  - Accept options as presented
- Expand the 06-StringsAndLists project that appears in Package Explorer (on the left-hand-side)
  - Browse the modules.
  - Let us do the exercise in the **1-daysOfYear.py** module

[Hidden] Solution

```
# Calculate day of year for a given date in a non leap year
months = ["jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec"]
length = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
m = input("Enter month name (3-letters, lowercase): ")[:-1] or .strip()
d = int(input("Enter the day of the month: "))

# Find out where in list of months this month falls
indx = months.index(m)
daysOfYr = 0
for i in range(indx):
    daysOfYr = daysOfYr + length[i]

daysOfYr = daysOfYr + d
m = m[0].upper() + m[1:]
print(m, d, "is day", daysOfYr, "of this year.")
```

Strings (character strings)

- String literals (constants):
- `"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." """`

Operating on Strings

Operations/Methods	What does each of these operation/method do?
s1 + s2	Concatenates two strings e.g. "xyz" + "abc"
*s * <int>*	Replicates string s <int> times e.g. "xyz" * 4
s.capetalize()	Copy of s with only 1st letter capitalized
s.lower()	Copy of s with all lower case characters
s.reverse()	Copy of s will all characters reversed
s.split()	Split s into a list of substrings

Some more string methods

Methods	What does each of these operation/method do?
s.count(sub)	Count the number of occurrences of sub in s
s.find(sub)	Find first position where sub occurs in s
s.title()	Copy of s with first character of each word capitalized
s.replace(old, new)	Replace all occurrences of old in s with new
s.lstrip()	Copy of s with leading white space removed
s.join(list)	Concatenate list into a string, using s as the separator

Practice with 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')

Q4-5

Strings are immutable sequences

- Lists are mutable:

```
colors = ["red", "white", "blue"]
colors[1] = "grey"
colors.append("cyan")
```

OK

colors becomes
["red", "grey", "blue"] then
["red", "grey", "blue". "cyan"]

- A string is an **immutable** sequence of characters

```
>>> building = "Taj Mahal"
>>> building[2]
>>> building[1:4]
>>> building[4] = "B"
```

NOT OK.
Gives an error message when executed.

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** creates a string from a list
  - `'#'.join(myList)`
  - `'Who#is#rich?#He#who#is#content.#Who#is#content?#Nobody.'`
- What is the value of `myList[0][2]`?
- Do exercise in **2-practiceWithStringsAndLists** module

Getting a string from the user

```
>>> name = input('Enter your name:')
Enter your name:John
>>> name
'John'
>>>
```

Q9 take a break

String Representation

- Computer stores 0s and 1s
  - Numbers stored as 0s and 1s
  - What about text?
- Text also stored as 0s and 1s
  - Each character has a code number
  - Strings are sequences of characters
  - Strings are stored as sequences of code numbers
  - Does it matter what code numbers we use?
- Translating: ord(<char>) chr(<int>)

Q10-11

Consistent String Encodings

- Needed to share data between computers, also between computers and display devices
- Examples:
  - ASCII—American Standard Code for Info. Interchange
    - "Ask-ee"
    - Standard US keyboard characters plus "control codes"
    - 8 bits per character
  - Extended ASCII encodings (8 bits)
    - Add various international characters
  - Unicode (16+ bits)
    - Tens of thousands of characters
    - Nearly every written language known

Q12

String Formatting

- Allows us to format complex output for display
  - It treats a string as a template with slots --- {}
  - Provided values are plugged into each slot
  - Uses a built-in method, format(), that takes values to plug into each slot
  - <template-string>.format(<values>)
- What does each slot look like?
  - {<index>:<format-specifier>}
  - <index> tells which of the parameters is inserted in slot
  - <format-specifier> describes how this slot will be formatted

Format Specifiers

- Syntax:
  - %<width>.<precision><typeChar>
- Width gives total spaces to use
  - 0 (or width omitted) means as many as needed
  - 0*n* means pad with leading 0s to *n* **total** spaces
  - -*n* means "left justify" in the *n* spaces
- Precision gives digits after decimal point, rounding if needed.
- TypeChar is:
  - f for float, s for string, or d for decimal (i.e., int) [can also use i]
- Note: this RETURNS a string that we can print
  - Or write to a file using write(string), as you'll need to do on the homework 6 assignment (HW6)

Q10-11 submit quiz

Begin HW6

- Although you do not have a reading assignment and Angel quiz, you are strongly encouraged to begin working on your homework early.