

Policies & Material

Questions

- modular design
- working with files
- object-oriented programming
- testing, exceptions, complexity
- GUI design and implementation

Policies & Material

Questions

- modular design
- working with files
- object-oriented programming
- testing, exceptions, complexity
- GUI design and implementation

MCS 260 Lecture 33
Introduction to Computer Science
Jan Verschelde, 12 November 2007

Questions

modular design
working with files
object-oriented programming
testing, exceptions, complexity
GUI design and implementation

Policies and Material

second midterm on Wednesday 14 November

The second midterm will be open book.

Calculators or laptop computers are not allowed.

The policy on skipping exams is: If an exam is missed, then greater weight will be placed on the final exam, especially on the material covered on the missing exam.

Skipping this midterm will make an application for an incomplete at the end of the semester very difficult.

Focus is on lectures 18 to 32, roughly divided in five:

(1) modules; (2) files; (3) OOP;

(4) testing, exceptions, complexity;

and (5) GUIs.

Questions

modular design
working with files
object-oriented programming
testing, exceptions, complexity
GUI design and implementation

Policies and Material

second midterm on Wednesday 14 November

The second midterm will be open book.

Calculators or laptop computers are not allowed.

The policy on skipping exams is: If an exam is missed, then greater weight will be placed on the final exam, especially on the material covered on the missing exam.

Skipping this midterm will make an application for an incomplete at the end of the semester very difficult.

Focus is on lectures 18 to 32, roughly divided in five:

(1) modules; (2) files; (3) OOP;

(4) testing, exceptions, complexity;

and (5) GUIs.

Policies and Material

second midterm on Wednesday 14 November

The second midterm will be open book.

Calculators or laptop computers are not allowed.

The policy on skipping exams is: If an exam is missed, then greater weight will be placed on the final exam, especially on the material covered on the missing exam.

Skipping this midterm will make an application for an incomplete at the end of the semester very difficult.

Focus is on lectures 18 to 32, roughly divided in five:

(1) modules; (2) files; (3) OOP;

(4) testing, exceptions, complexity;

and (5) GUIs.

Policies & Material

Policies & Material

Questions

Questions

modular design

modular design

working with files

working with files

object-oriented programming

object-oriented
programming

testing, exceptions, complexity

testing, exceptions,
complexity

GUI design and implementation

GUI design and
implementation

1. Design the modular structure

Design the modular structure of an online system to help planning a trip using public transport, for example with the CTA.

What is at the bottom of the program?

Describe the modules and their relations.

Refer to the three key principles of a good design to explain why you choose this design.

Start by thinking about only one subway or bus line.

Hint: the system map is at the bottom of the program.
Review our first modular design of the library manager.

1. Design the modular structure

Design the modular structure of an online system to help planning a trip using public transport, for example with the CTA.

What is at the bottom of the program?

Describe the modules and their relations.

Refer to the three key principles of a good design to explain why you choose this design.

Start by thinking about only one subway or bus line.

Hint: the system map is at the bottom of the program.
Review our first modular design of the library manager.

2. Modular design principles

One of the modular design principles we discussed is *low coupling and high cohesion*.

Explain what this means and give examples indicating which properties hold.

Why is this a desirable property of the design of large programs?

Answer: Low coupling means that modules have few dependencies on other modules. High cohesion means that related functions or functions that highly interact with other belong to the same module.

Typically, we would concentrate functions that communicate with the user in the same module.

Low coupling and high cohesion allows modules to be developed and tested independently of the rest of the program.

2. Modular design principles

One of the modular design principles we discussed is *low coupling and high cohesion*.

Explain what this means and give examples indicating which properties hold.

Why is this a desirable property of the design of large programs?

Answer: Low coupling means that modules have few dependencies on other modules. High cohesion means that related functions or functions that highly interact with other belong to the same module.

Typically, we would concentrate functions that communicate with the user in the same module.

Low coupling and high cohesion allows modules to be developed and tested independently of the rest of the program.

- modular design
- working with files
- object-oriented programming
- testing, exceptions, complexity
- GUI design and implementation

3. Software development models

Compare the waterfall model to the spiral model of software development.

Enumerate at least two relative advantages and disadvantages of each model.

Which type of model would fit what type of company organization best?

Answer: The waterfall model has three distinctive phases before code gets written, whereas the spiral model strives to get a first prototype out quickly. So the spiral model gets results more quickly and gives feedback sooner. The waterfall model will produce a more detailed and complete specification than the spiral model. The waterfall model benefits large organizations where analysis is done separately from coding. In smaller organizations, where the same people often take on many different tasks, the spiral model is more natural.

3. Software development models

Compare the waterfall model to the spiral model of software development.

Enumerate at least two relative advantages and disadvantages of each model.

Which type of model would fit what type of company organization best?

Answer: The waterfall model has three distinctive phases before code gets written, whereas the spiral model strives to get a first prototype out quickly. So the spiral model gets results more quickly and gives feedback sooner. The waterfall model will produce a more detailed and complete specification than the spiral model. The waterfall model benefits large organizations where analysis is done separately from coding. In smaller organizations, where the same people often take on many different tasks, the spiral model is more natural.

Policies & Material

Policies & Material

Questions

modular design
working with files
object-oriented programming
testing, exceptions, complexity
GUI design and implementation

Questions

modular design
working with files
object-oriented programming
testing, exceptions, complexity
GUI design and implementation

4. word removal from file

Consider the problem of removing all occurrences of one word from a text file.

1. Describe the design of a program to remove a user given word for a text file whose name is given by the user as well. View the file as a sequence of characters. What data structures will you use? Show the control flow of the program using a flowchart.
2. Give the Python code for a program to remove all occurrences of a user given word.

design of word removal from file

part of the answer to question 4

The data structure we will use is a buffer to hold exactly as many characters as the length of the word we want to delete from file.

The writing to file will only start if the buffer is full, and of course if the content of the buffer is distinct from the word we want to delete.

The buffer is managed as a FIFO queue: as the newest character gets added to the end, the oldest character leaves the buffer. The character that leaves the buffer gets written to file.

So the key ingredient in the solution to this problem is the function to update the buffer.

updating buffer in word removal

another part of the answer to question 4

```
def add_to_buffer(file,n,b,c):
    """
    Adds c to a n-letter buffer b,
    writes letter that drops off to file.
    """
    B = b
    if len(b) == n:
        file.write(B[0])
        for i in range(0,n-1):
            B[i] = B[i+1]
        B[n-1] = c
    else:
        B.append(c)
    return B
```

The main program makes repeated calls of the type
`b = add_to_buffer(outfile,n,b,c)` for every new
character `c` read from the input file.

5. files with structure

The Unix command `cal` produces a calendar as (`$` is the command prompt)

```
$ cal 11 2007
    November 2007
Su Mo Tu We Th Fr Sa
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

Note that the last line is blank. Assume that this output of `cal 11 2007` is an input file for a Python program that should then write `November 2007 has 30 days`.

1. Draw the flowchart for the algorithm.
2. Give the Python program to implement the algorithm.

Python code for question 5

second part of the answer

```
name = raw_input('file with month : ')
file = open(name, 'r')
month = ''
year = ''
max = 0
while True:
    s = file.readline()
    if s == '': break
    L = s.split()
    if month == '':
        month = L[0]
        year = L[1]
    elif max == 0:
        max = 1
    elif len(L) > 0:
        if int(L[len(L)-1]) > max:
            max = int(L[len(L)-1])
print month + ' ' + year + ' has ' + \
    str(max) + ' days'
```

6. file permissions

Explain what `-rwxr-x--x` is and what it means.

Answer: `-rwxr-x--x` is an access permission code for a file on Unix.

The following operations on the file are permitted:

1. the owner can read, write, and execute;
2. group members can read and execute;
3. all others can only execute.

6. file permissions

Explain what `-rwxr-x--x` is and what it means.

Answer: `-rwxr-x--x` is an access permission code for a file on Unix.

The following operations on the file are permitted:

1. the owner can read, write, and execute;
2. group members can read and execute;
3. all others can only execute.

Policies & Material

Questions

modular design
working with files
object-oriented programming
testing, exceptions, complexity
GUI design and implementation

7. OO design and programming

Suppose we want to administer a small online school. All administrative functions are taken care of by a program.

1. Use UML, defining class and use case diagrams to model the administrative programs. Students enroll in courses taught by faculty. Faculty administer their courses: create, delete and enter grades.
2. Give Python code to define the classes, i.e.: list the data and functional attributes.

Hint: review the object oriented design of the library management system. Instead of books we have courses, instead of patrons and librarians, we have students and faculty.

7. OO design and programming

Suppose we want to administer a small online school. All administrative functions are taken care of by a program.

1. Use UML, defining class and use case diagrams to model the administrative programs. Students enroll in courses taught by faculty. Faculty administer their courses: create, delete and enter grades.
2. Give Python code to define the classes, i.e.: list the data and functional attributes.

Hint: review the object oriented design of the library management system. Instead of books we have courses, instead of patrons and librarians, we have students and faculty.

8. Inheritance

A library contains music CDs and books.

Describe a class `Item` that defines data attributes common to both music CDs and books.

For the classes `CD` and `Book` inheriting from `Item`, define additional data attributes for both of them.

Partial Answer: Data common to both CDs and books are title and author. These attributes belong then to the class `Item`.

About a CD, we may want to store the number of tracks and the total duration. These are two examples of object data attributes for the class `CD`.

Object data attributes to be added to the class `Book` are for example the edition and the total number of pages.

8. Inheritance

A library contains music CDs and books.

Describe a class `Item` that defines data attributes common to both music CDs and books.

For the classes `CD` and `Book` inheriting from `Item`, define additional data attributes for both of them.

Partial Answer: Data common to both CDs and books are title and author. These attributes belong then to the class `Item`.

About a CD, we may want to store the number of tracks and the total duration. These are two examples of object data attributes for the class `CD`.

Object data attributes to be added to the class `Book` are for example the edition and the total number of pages.

Review II

MCS 260 L-33

12 November 2007

Policies & Material

Policies & Material

Questions

modular design
working with files
object-oriented
programming

**testing, exceptions,
complexity**

GUI design and
implementation

Questions

modular design
working with files
object-oriented programming
testing, exceptions, complexity
GUI design and implementation

9. What is a bug?

Give an example of rule #4 of a bug:

“The software doesn't something that the specification doesn't mention, but should.”

Explain why the rule applies to your example.

Answer: Consider for example a calculator (or software that does what a handheld calculator does).

The calculator does not do addition and also the manual does not mention that it does addition. However, we expect from any calculator that it does addition, so therefore the absence of addition is a bug.

9. What is a bug?

Give an example of rule #4 of a bug:

“The software doesn't something that the specification doesn't mention, but should.”

Explain why the rule applies to your example.

Answer: Consider for example a calculator (or software that does what a handheld calculator does).

The calculator does not do addition and also the manual does not mention that it does addition. However, we expect from any calculator that it does addition, so therefore the absence of addition is a bug.

10. exception handling

The command `uname` returns information about the OS on Unix, Linux, and MacOS X systems.

Write a function that returns the result of `uname`, and provide a handler in case we exceptionally use another OS.

Answer:

```
def MyUname():
    "executes uname and returns result"
    import os
    try:
        os.system('uname')
    except OSError:
        print 'windows'
```

10. exception handling

The command `uname` returns information about the OS on Unix, Linux, and MacOS X systems.

Write a function that returns the result of `uname`, and provide a handler in case we exceptionally use another OS.

Answer:

```
def MyUname():
    "executes uname and returns result"
    import os
    try:
        os.system('uname')
    except OSError:
        print 'windows'
```

modular design
working with files
object-oriented
programming

**testing, exceptions,
complexity**

GUI design and
implementation

11. Software Testing

Explain the differences between static blackbox and dynamic whitebox testing.

Answer: In static blackbox testing we do not execute the program and we do not see the code either, whereas in dynamic blackbox testing we will execute programs and review the code.

modular design
working with files
object-oriented
programming

**testing, exceptions,
complexity**

GUI design and
implementation

11. Software Testing

Explain the differences between static blackbox and dynamic whitebox testing.

Answer: In static blackbox testing we do not execute the program and we do not see the code either, whereas in dynamic blackbox testing we will execute programs and review the code.

12. counting operations

Consider the inner product of two lists of numbers, A and B , both of length n , defined as

$$A[0]*B[0] + A[1]*B[1] + \dots \\ \dots + A[n-1]*B[n-1].$$

Count the number of arithmetical operations.

Show that it is $O(n)$.

Answer: consider first the case $n = 3$:

$$A[0]*B[0] + A[1]*B[1] + A[2]*B[2].$$

For $n = 2$, we count 2 + and 3 *.

For general n , we count $n - 1$ + and n *.

In total, we have $2n - 1$ operations.

As $2n - 1 \leq 2n$, the cost is $O(n)$.

12. counting operations

Consider the inner product of two lists of numbers, A and B , both of length n , defined as

$$A[0]*B[0] + A[1]*B[1] + \dots \\ \dots + A[n-1]*B[n-1].$$

Count the number of arithmetical operations.

Show that it is $O(n)$.

Answer: consider first the case $n = 3$:

$$A[0]*B[0] + A[1]*B[1] + A[2]*B[2].$$

For $n = 2$, we count 2 + and 3 *.

For general n , we count $n - 1$ + and n *.

In total, we have $2n - 1$ operations.

As $2n - 1 \leq 2n$, the cost is $O(n)$.

Policies & Material

Questions

modular design
working with files
object-oriented programming
testing, exceptions, complexity
GUI design and implementation

13. different buttons

In what sense are the widgets

Radiobutton, Checkbutton, and Scale similar?

What are their differences?

Give three examples of their most appropriate use.

Answer: the similarity is that all three widgets are used to enter values for parameters. They each offer different ranges. A Radiobutton may be used to increase or decrease a default value by a fixed amount.

A Checkbutton offers only two choices. While the resolution in a Scale might coincide with the increments in a Radiobutton, a Scale has lower and upper limits.

Three examples of good use for each:

1. Radiobutton: to set a certain finite amount;
2. Checkbutton: to choose gender;
3. Scale: to determine a value in an interval.

13. different buttons

In what sense are the widgets

Radiobutton, Checkbutton, and Scale similar?

What are their differences?

Give three examples of their most appropriate use.

Answer: the similarity is that all three widgets are used to enter values for parameters. They each offer different ranges. A Radiobutton may be used to increase or decrease a default value by a fixed amount.

A Checkbutton offers only two choices. While the resolution in a Scale might coincide with the increments in a Radiobutton, a Scale has lower and upper limits.

Three examples of good use for each:

1. Radiobutton: to set a certain finite amount;
2. Checkbutton: to choose gender;
3. Scale: to determine a value in an interval.

14. design a GUI

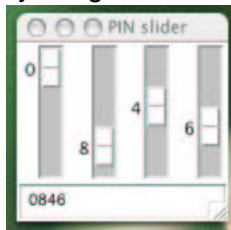
Consider a system where entry is determined by a four digit access code (such as 9812).

1. Design a GUI where the user can *only* enter a four digit access code. By *only* we mean that (1) only 4 digits can be entered; and (2) only digits (i.e.: numbers in the range 0,1,...,9) are permitted. What kind of widgets will you use?
2. Write the constructor function in the object oriented implementation of this GUI.
3. Give the functional attributes in the class that implements this GUI.

Slide in your PIN

answer to question 14

The GUI to admit only 4 digit PINs is



The layout of the GUI is as follows:

1. four scales from 0 to 9 and resolution = 1;
2. one entry field to show the PIN number.

The object data attributes are the four digits of the PIN, instances of `IntVar` and set by the scales.

There is one function: `ShowPin` that will take the four digits and fill in the Entry widget.

Questions

modular design
working with files
object-oriented programming
testing, exceptions, complexity
GUI design and implementation

Class GuiPin

answer to question 14 continued

```
from Tkinter import *

class GuiPin():
    """
    Four scales to enter a 4-digit number.
    """
    def __init__(self,wdw):
        "defines layout of the GUI"

    def ShowPin(self,v):
        "puts the PIN in the entry widget"

def main():
    top = Tk()
    show = GuiPin(top)
    top.mainloop()

if __name__ == "__main__": main()
```

15. a GUI to order food

Consider an ordering system in a fastfood restaurant selling burgers, hotdogs, and pizza slices. French fries are sold in small, medium, and large quantities.

1. Design a GUI to make ordering easy.
The user should first see the total price of the order, before confirming the order.
Define the layout and the actions of the GUI.
2. Give the object oriented design of the GUI, defining the data and functional attributes of the class.
Write proper documentation strings.
3. Give Python code to implement this ordering system.

Answer: Kathy's fastfood restaurant was shown in the lab sessions.