

# SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2017

## 20 Midterm Preparation

Dr. Spencer Smith

Faculty of Engineering, McMaster University

February 17, 2017



# Midterm Preparation

- Administrative details
- Topics on midterm
- Sample midterm questions

# Administrative Details

- Bookstore returning textbooks in first few weeks of March
- Assignment 1 grades now available in Avenue or repo
- Assignment 2
  - ▶ Files due by 11:59 pm Feb 19
  - ▶ File automatically sent to partners on Feb 20
  - ▶ Lab report due by 11:59 pm Feb 27
- Midterm exam
  - ▶ March 1, 7:00 pm, TSH/120
  - ▶ 90 minute duration
  - ▶ Multiple choice - 30–40 questions?
  - ▶ Open book (any paper)

# Midterm Advice

- Some questions will take longer to answer
- Read the questions carefully
- Inspect code and/or mathematics carefully
- Eliminate alternatives
- Take your time
- Questions are not “tricks,” but, just like code or mathematics you need to look and think carefully
- Selecting the correct answer will not depend on esoteric Python syntax
- The correct answer is the best answer
- There will be no (intentional) syntax or mathematical errors

# Topics on Midterm

- Everything we have covered in class and tutorial up to today
- Partial list provided here
- Ghezzi Chapters 1 to 4 (inclusive)
- Hoffmann and Strooper
  - ▶ Explicitly covered
    - ▶ Chapter 3
    - ▶ Chapter 6
    - ▶ Chapter 7
  - ▶ Implicitly covered
    - ▶ Chapter 1
    - ▶ Chapter 2

# Topics on Midterm Continued

- Introduction to software development/software engineering profession
  - ▶ What is software engineering?
  - ▶ PEO?
  - ▶ Therac-25 and other failures
  - ▶ Software development processes
  - ▶ “Faked” rational design process
  - ▶ Software documentation
- Software quality
  - ▶ Definition of software quality
    - ▶ Internal versus external
    - ▶ Product versus process
  - ▶ Software qualities (correctness, reliability, reusability etc.)
  - ▶ Relationship between qualities
  - ▶ Measurement of quality

# Topics on Midterm Continued

- Software engineering principles
  - ▶ Software engineering knowledge units
  - ▶ Key principles
    - ▶ Rigour
    - ▶ Formality
    - ▶ Separation of concerns
    - ▶ Modularity
    - ▶ Abstraction
    - ▶ Anticipation of change
    - ▶ Generality
    - ▶ Incrementality
- Introduction to modules
  - ▶ What is design?
  - ▶ Program families
  - ▶ Interface versus implementation
  - ▶ Information hiding
  - ▶ Examples of modules (record, library, ...)

# Topics on Midterm Continued

- Mathematical fundamentals
- MIS
  - ▶ Uses
  - ▶ Syntax
  - ▶ Semantics
    - ▶ State variables
    - ▶ State invariants
    - ▶ Assumptions
    - ▶ Access routine semantics
    - ▶ Local functions
    - ▶ Local types
    - ▶ Local constants
    - ▶ Considerations
- Specification Abstract Objects
- Specification of ADTs
- Generic MIS



# Topics on Midterm Continued

- Implementation of abstract objects in Python
- Implementation of abstract data types in Python
- Set, sequence, tuple interface design idioms
- Assumptions versus exceptions
- Object oriented design
  - ▶ Inheritance
  - ▶ Polymorphism
  - ▶ Dynamic binding
- Introduction to UML
- Cohesion and coupling

# Topics on Midterm Continued

- Quality criteria for MIS
  - ▶ Consistent
  - ▶ Essential
  - ▶ General
  - ▶ Implementation independent
  - ▶ Minimal
  - ▶ Opaque
- Modular decomposition
- Techniques for design for change
- Relation between modules
- Uses relation
- DAG, hierarchy, tree
- Cohesion and coupling, fan in versus fan out
- Is\_COMPONENT\_OF relation

# Topics on Midterm Continued

- Module guide
- Behaviour, software decision and hardware hiding modules
- Typical modules
- Modules with external interaction
  - ▶ Environment variables
  - ▶ Changing the state of another module
  - ▶ Hidden interactions
- Functional programming
  - ▶ List comprehensions
  - ▶ Map
  - ▶ Lambda functions
  - ▶ Partial functions
  - ▶ Filter
  - ▶ Reduce

# Topics on from Tutorials

- Doxygen
- Make
- LaTeX
- git
- PyUnit

# Python

The Python programming language fully supports the principle of information hiding. Is this statement True or False?

- A. True
- B. False

## Sample Question

A module without any state variables and without any output cannot serve any practical purpose. Is this statement True or False?

- A. True
- B. False

## Sample Question

If a specification is formal, it still might not be possible to implement it in code. Is this statement True or False?

- A. True
- B. False

# Question

## Module

MAX\_SIZE = 100

Routine name	In	Out	Exceptions
seq_init			
seq_add	integer, integer		POS
seq_del	integer		POS
seq_setval	integer, integer		POS
seq_getval	integer	integer	POS
seq_size		integer	
seq_isFull		boolean	

The interface for this sequence module is essential; that is, unnecessary access programs are omitted.

- A. True
- B. False



## Sample Question

seq\_add( $i, p$ ):

- transition:  $(|s| = \text{MAX\_SIZE} \Rightarrow s = s \mid |s| < \text{MAX\_SIZE} \Rightarrow s := s[0..i-1] \parallel \langle p \rangle \parallel s[i..|s|-1])$
- exception:  $\text{exc} := (i \notin [0..|s|] \Rightarrow \text{POS})$

Consider the above specification for the add access program for a sequential abstract object with state variable  $s$  of type sequence of int. This specification ensures that the state invariant is satisfied. Is this statement True or False.

- A. True
- B. False

## Sample Question

```
class Seq:
    MAX_SIZE = 100
    s = []
    @staticmethod
    def init():
        Seq.s = []
```

Consider the partial implementation of the abstract object for sequence. The implementation is incorrect, since `init` should follow the convention of `__init__` and `init` should take the implicit parameter `self`. Is this statement True or False?

- A. True
- B. False

## Sample Question

$s$ : sequence of integer

mystery( $j$ ):

- output:  $\text{out} := + (i : \mathbb{N} \mid i \in s \wedge i \geq j : 1)$

What does the mystery access program return?

- A. The sum of the entries in  $s$  that are greater or equal to  $j$
- B. The sum of the entries in  $s$  that are greater than  $j$
- C. A count of the number of entries in  $s$  that are greater or equal to  $j$
- D. A count of the number of entries in  $s$  that are greater than  $j$
- E. True if there exists an entry in  $s$  that is greater than  $j$ ,  
False otherwise

## Sample Question

s: sequence of integer

mystery(j):

- output:  $\text{out} := + (i : \mathbb{N} | i \in [0..|s| - 1] \wedge s[i] \geq j : 1)$

Which of the following implements mystery(j) in Python?

- A. `reduce(lambda a, b: a+b, [1 for i in range(len(s)) if s[i] >= j], 0)`
- B. `reduce(lambda a, b: a+b, [s[i] for i in range(len(s)) if s[i] >= j], 0)`
- C. `reduce(lambda a, b: a or b, [(i >= j) for i in s], False)`
- D. `reduce(lambda a, b: a or b, [(i >= j) for i in s], True)`

## Sample Question

```
from functools import partial
def power(base, exponent):
    return base ** exponent
cube = partial(power, exponent=3)
```

Alternatively, a version of cube that calculates the same results can be defined via:

- A. 

```
def cube(b):
    return power(b, 3)
```
- B. 

```
cube = lambda x: power(x, 3)
```
- C. Both A and B
- D. Neither A or B