LECTURER: TAI LE QUY

ALGORITHMS, DATA STRUCTURES, AND PROGRAMMING LANGUAGES

TOPIC OUTLINE

TOPIC OUTLINE

UNIT 5

MEASURING PROGRAMS

STUDY GOALS

- Apply type inference mechanisms.
- Understand tools to generate documentation.
- Understand compiler optimization techniques.
- Know about tools for code coverage analysis.
- Understand the principles of unit and integration testing and apply a selection of them.
- Apply heap analysis tools in Python.

EXPLAIN SIMPLY

1. Explain what you understand by type inferencing with examples from the meta language (ML) programming language.
2. Briefly describe five different ways in which a compiler optimizes code.
3. Explain what the *Memory Profiler* in Python is and how it is used.

ML Programming Language

- ML is primarily a functional programming language.
- It also has support for imperative style of programming.
- It is strongly typed.
- All types are statically inferred.
- Type declarations are not required if the types can be derived unambiguously.

ML Example

Definitions are all equivalent and all produce the correct result, whenever the type (real) of at least one of x, y or the function is specified.
The type inference mechanism infers the missing types as real.

```
fun f(x:real, y:real):real = x
+ y;
```

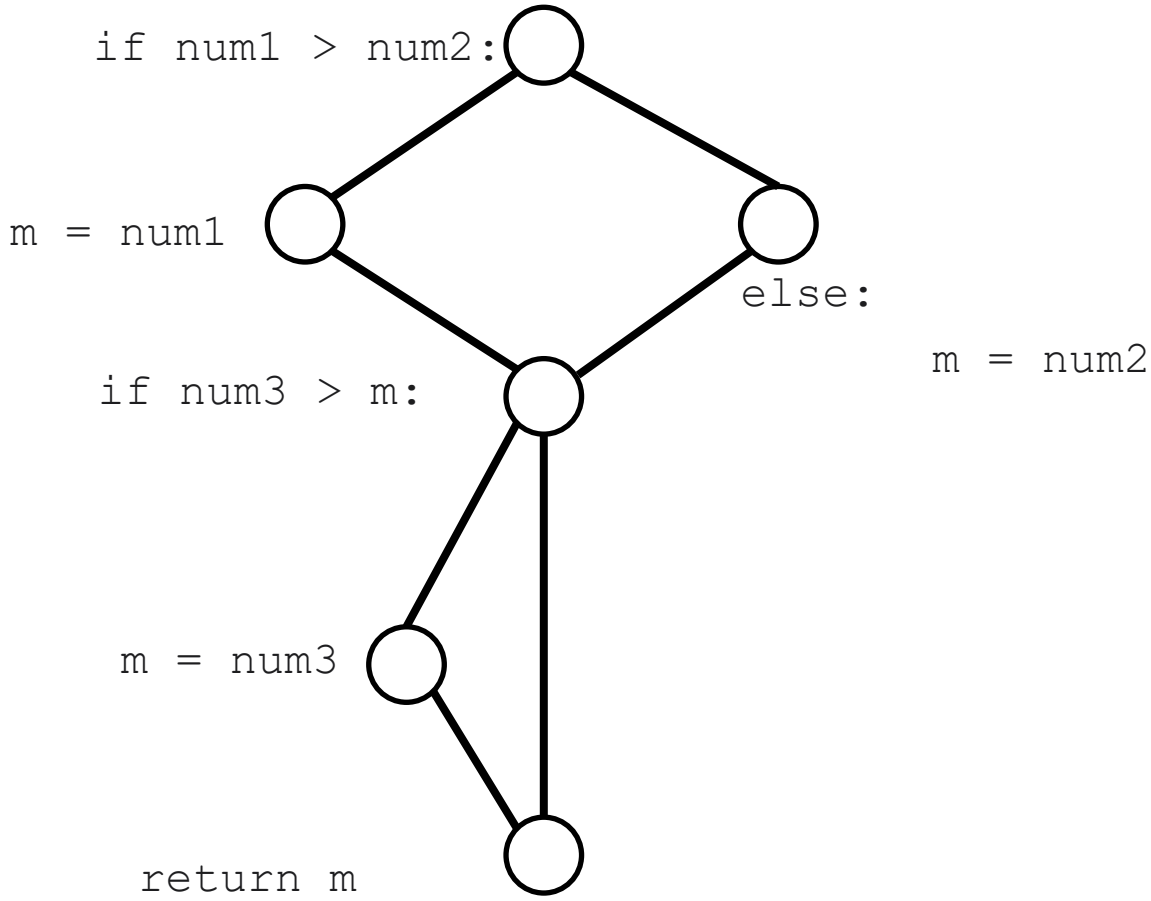
```
fun semiperimeter7(width:real, height:real):real = width + height;
fun semiperimeter6(width:real, height:real) = width + height;
fun semiperimeter5(width:real, height):real = width + height;
fun semiperimeter4(width:real, height) = width + height;
fun semiperimeter3(width, height:real):real = width + height;
fun semiperimeter2(width, height:real) = width + height;
fun semiperimeter1(width, height):real = width + height;
```

All these definitions are equivalent and produce the same correct result.
If none of the types are specified, all the types default to integer and the program reports an error when invoked with real parameters.

```
fun semiPerimeter0(width, height) = width + height;
```

CYCLOMATIC COMPLEXITY

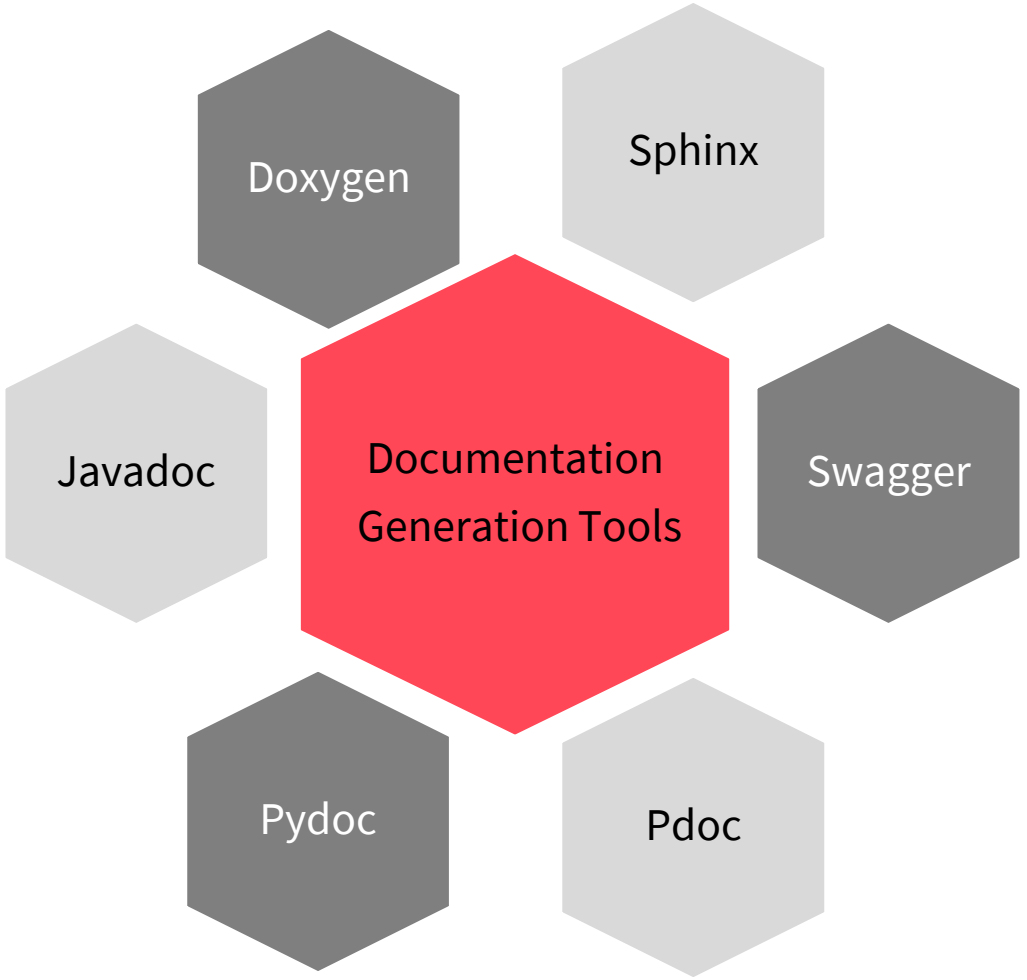
- **Cyclomatic complexity** measures the complexity in a program.
- It tries to quantify the testability and maintainability of software.
- Measures the complexity of the control structure of the program.
- It is the number of linearly independent paths through the **control flow graph** of the program.
- For a graph with v vertices and e edges, CC(G) = e – v + 2



CFG with Cyclomatic Complexity = 7 − 6 + 2 = 3

DOCUMENTATION GENERATION

- Documentation assists in better understanding of the code.
- Poor content or ambiguous information is often a problem.
- Tools for automatic generation and recommendation of documentation have been developed.
- There are tools based on text summarization algorithms.
- They create summaries from classes, methods, bug reports, code changes, unit tests.



PYTHON DOCUMENTATION BEST PRACTICES

DOCUMENTING CLASSES

A class level docstring should describe purpose and operations.

DOCUMENTING MODULES

Each module should start with a top-level docstring which outlines the purpose of the module.

DOCUMENTATION BEST PRACTICES USING PYTHON DOCSTRINGS

DOCUMENTING FUNCTIONS

Explanatory entries for function arguments and return values and any special behaviors.

COMPILER OPTIMIZATION

MAKE SMALL FUNCTIONS INLINE
Replace function calls with code for the function.

MOVING REPETITIVE COMPUTATIONS OUTSIDE LOOPS
Such computations can be done only once outside the loop.

LOOP UNROLLING
Repeat the code in the loop.

COMPILER OPTIMIZATION
Program transformation techniques to improve the code

ELIMINATE COMMON SUBEXPRESSIONS
The common subexpression needs to be computed only once.

ELIMINATE REDUNDANT STORES
An assignment operation which is redundant may be removed.

ELIMINATE UNREACHABLE CODE
Such code serves no useful purpose.

REDUCTION IN STRENGTH
Replace operations with ones which are more efficient.

CODE COVERAGE METRICS

May be applied to a file, module, library or a system.

#STATEMENTS OR LINES OF CODE
Commonly used measure

#BLOCKS
A block is a sequence of non-branching instructions; together considered a single unit.

CODE COVERAGE
Software metric that tries to quantify to what extent a software is verified. Counted at various levels of granularity.

#CLASSES, FUNCTIONS, BRANCHES

#LOOPS
Count how many loops are never executed, executed only once or executed more than once.

UNIT TESTS: BEST PRACTICES

TESTS CAUSING INPUT BUFFERS TO OVERFLOW

TESTS HAVING SAME SEQUENCE OF INPUTS SEVERAL TIMES

TESTS LEADING TO GENERATION OF ALL POSSIBLE ERROR MESSAGES

UNIT TESTS
Accepted best practices

TESTS RESULTING IN INVALID OUTPUTS TO BE GENERATED

TESTS GENERATING EXTREMELY LARGE OR EXTREMELY SMALL NUMERIC OUTPUTS

TESTING INTERFACES IN NTEGRATION TESTING

SHARED MEMORY INTERFACES

Interfaces in which components share a block of memory.

PARAMETER INTERFACES

Components exchange data or function references through these interfaces.

INTEGRATION TESTING

In software testing, after individual objects have been tested, interfaces are tested.

PROCEDURAL INTERFACE

One component encapsulates procedures or functions which are called by other components.

MESSAGE PASSING INTERFACE

Interfaces in which one component requests a service by passing a message.

The Memory Profiler in Python

- Python library psutil monitors and retrieves information on running processes and system utilization.
- The Memory profiler is a Python module built on top of psutil.
- It monitors the memory usage of a process.
- Generates line-by-line analysis of the memory consumption.

Memory Profiler Usage

Command line invocation:

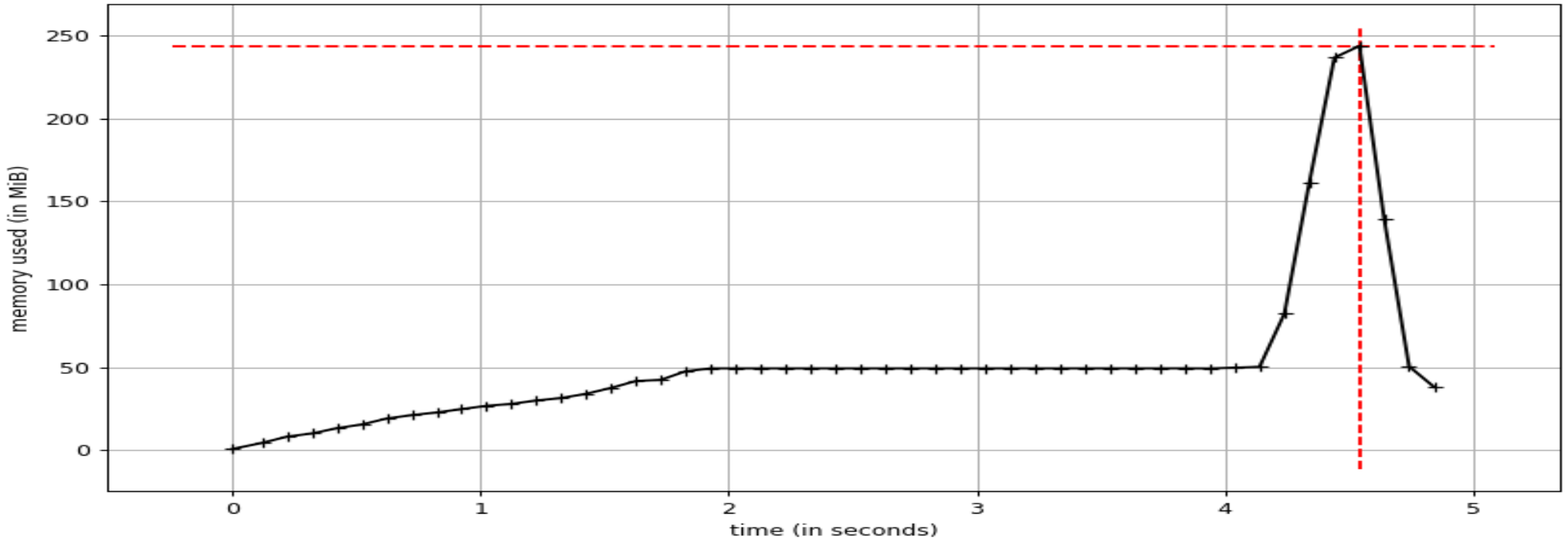
```
python -m memory_profiler
filename.py
```

The functions being profiled can be marked with the decorator @profile

```
from memory_profiler
import profile
@ profile
def A():
```

To generate and plot the memory usage over time run:

```
mprof run filename.py
mprof plot
```



Source of the image: Coursebook DLBCSL01.

REVIEW STUDY GOALS

- Apply type inference mechanisms.
- Understand tools to generate documentation.
- Understand compiler optimization techniques.
- Know about tools for code coverage analysis.
- Understand the principles of unit and integration testing and apply a selection of them.
- Apply heap analysis tools in Phyton.

SESSION 4

TRANSFER TASK

TRANSFER TASKS

1. The goal of this exercise is to measure the code coverage of a Python program using the Coverage.py tool.
   (a) Write a Python program which reads in an integer and prints whether the integer is negative, zero or positive. Write your code using the `if-elif-else` construct.
   (b) Write a test program to measure coverage by first including a test case in which the input integer is negative and running the Coverage.py tool.
   (c) Repeat the exercise by first adding a test case for the input being zero and then again for the input being positive.
   (d) Describe and explain your observations.

TRANSFER TASK
PRESENTATION OF THE RESULTS

Please present your results.

The results will be discussed in plenary.

1. If a control flow graph with five vertices has a cyclomatic complexity of two, how many edges does it have?
   a) Four
   b) Five
   c) Six
   d) Seven

2. What do we call the degree to which test suite exercises a software system?
   a) Code coverage
   b) Code range
   c) Code content
   d) Code scope

3. In compiler optimization, what is the name of a technique that is applied to a small amount of code in a sliding window?
   a) Sliding optimization
   b) Window optimization
   c) Sliding window optimization
   d) Peephole optimization

LIST OF SOURCES

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compilers: Principles, Techniques, and Tools* (2nd edition). Addison-Wesley.
Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson.