

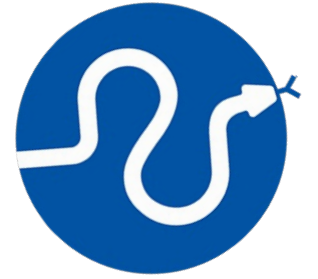
STATS 507

Data Analysis in Python

Week7: SciPy and Midterm Recap

Dr. Xian Zhang

In-class practice



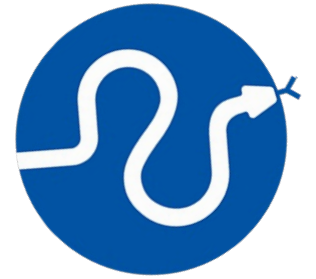
NumPy practice (Bonus in-class practice): **walking through the solution**

Come to my office hours if you want to discuss more about it!

Part 2: SciPy



1. SciPy as Scientific Python
2. Matplotlib for visualization



What is SciPy

SciPy: **Sci**entific **Py**thon

Designed and used for scientific and technical computing

SciPy is a library (collection) of **algorithms** and **mathematical** tools built to work with NumPy arrays.

- Mathematics, science, and engineering.
- Can handle operations and data analysis that goes beyond what basic Python or even NumPy can handle.

When to use SciPy

SciPy is organized into **subpackages** covering different scientific computing domains.

[Special functions](#) (`scipy.special`)

[Integration](#) (`scipy.integrate`)

[Optimization](#) (`scipy.optimize`)

[Interpolation](#) (`scipy.interpolate`)

[Fourier Transforms](#) (`scipy.fft`)

[Signal Processing](#) (`scipy.signal`)

[Linear Algebra](#) (`scipy.linalg`)

[Sparse Arrays](#) (`scipy.sparse`)

[Sparse eigenvalue problems with ARPACK](#)

[Compressed Sparse Graph Routines](#) (`scipy.sparse.csgraph`)

[Spatial data structures and algorithms](#) (`scipy.spatial`)

[Statistics](#) (`scipy.stats`)

[Multidimensional image processing](#) (`scipy.ndimage`)

[File IO](#) (`scipy.io`)

How to use SciPy: importing

Everything in the namespaces of SciPy submodules is **public**. In general in Python, it is recommended to make use of namespaces. For example:

```
my_matrix = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])
# Method 1: Import the entire SciPy package
import scipy
# Usage
eigenvalues = scipy.linalg.eigvals(matrix)
```

```
# Method 2: Import specific submodules
from scipy import special, linalg
# Usage
eigenvalues = linalg.eigvals(matrix)
```

```
# Method 3: Import specific functions
from scipy.linalg import eigvals
# Usage
eigenvalues = eigvals(matrix)
```

```
print(type(eigenvalues))
print(eigenvalues)

<class 'numpy.ndarray'>
[ 1.61168440e+01+0.j -1.11684397e+00+0.j -8.58274334e-16+0.j]
```

SciPy Linear Algebra

There is overlap in the functionality provided by the SciPy and NumPy submodules. However:

- Slightly different from `numpy.linalg`. Always uses BLAS/LAPACK support, which are often highly optimized for specific hardware so could be **faster**.
- Unless you don't want to add SciPy as a dependency to your NumPy program, **use `scipy.linalg` instead of `numpy.linalg`**

BLAS: basic linear algebra subprogram: <https://www.netlib.org/blas/>

LAPACK: linear algebra Package: <https://www.netlib.org/lapack>

<https://docs.scipy.org/doc/scipy/tutorial/linalg.html>

SciPy Linear Algebra

`scipy.linalg` contains all the functions in `numpy.linalg`

SciPy linear algebra could be **faster**.

<https://docs.scipy.org/doc/scipy/tutorial/linalg.html>

```
import numpy as np
import scipy.linalg as spla
import numpy.linalg as npla
import time
```

Solve a linear system: $Ax = b$

```
# Create a large matrix
```

```
n = 1000
```

```
A = np.random.rand(n, n)
```

```
b = np.random.rand(n)
```

```
# NumPy solve
```

```
start_time = time.time()
```

```
x_np = npla.solve(A, b)
```

```
np_time = time.time() - start_time
```

```
# SciPy solve
```

```
start_time = time.time()
```

```
x_sp = spla.solve(A, b)
```

```
sp_time = time.time() - start_time
```

```
print(f"NumPy solve time: {np_time:.6f} seconds")
```

```
print(f"SciPy solve time: {sp_time:.6f} seconds")
```

```
print(f"Speed difference: {np_time/sp_time:.2f}x")
```

```
NumPy solve time: 0.044730 seconds
```

```
SciPy solve time: 0.020878 seconds
```

```
Speed difference: 2.14x
```

SciPy Sparse

Sparse **matrix classes**: CSC, CSR, etc.

- functions to **identify** and **build** sparse matrices
- `sparse.linalg` module for sparse linear algebra
- `sparse.csgraph` for sparse graph routines

```
import numpy as np
from scipy import sparse
# 1. Sparse matrix classes: CSC, CSR
dense_matrix = np.array([
    [1, 0, 2],
    [0, 3, 4],
    [5, 6, 0]
])

# Create Compressed Sparse row matrix
csr_matrix = sparse.csr_matrix(dense_matrix)
print("CSR Matrix:")
print(type(csr_matrix))
print(csr_matrix)

CSR Matrix:
<class 'scipy.sparse._csr.csr_matrix'>
(0, 0)    1
(0, 2)    2
(1, 1)    3
(1, 2)    4
(2, 0)    5
(2, 1)    6
```

```
# sparse.linalg module for sparse linear algebra
# Solve a sparse linear system Ax = b
from scipy.sparse import linalg as spla
A = sparse.csr_matrix([[1, 2], [3, 0]])
b = np.array([1, 2])
x = spla.spsolve(A, b)
print("\nSolution to Ax = b:")
print(x)
```

Can be even more efficient
Since we can make use of
the structure

More subroutines

Scipy **Statistics**

- Distributions
- Functions (Mean, median, mode, variance)
- Hypothesis tests
- ...

More on this later!

Scipy **Signal**

- Continuous-time linear system
- Filtering
- ...

Scipy **IO**

- Methods for loading and saving data
- Matlab files
- Matrix Market files (sparse matrices)
- Wav files

In-class practice

On SciPy Optimization

Midterm Recap

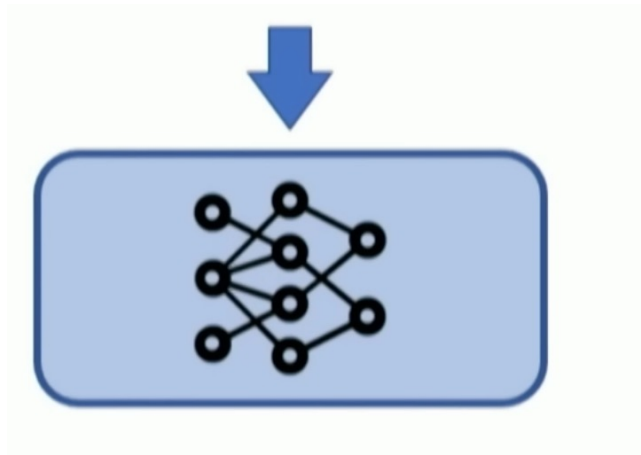
Recall our course goals

1. Establish a broad background in **Python** programming
2. Prepare you for the inevitable coding interview
3. Survey popular **tools** in academia/industry for data analysis
4. Learn how to read documentation and quickly get familiar with new tools.

The scope of midterm will be everything covered in lecture, in-class practice, HWs up to and including NumPy. The review session will just recap some of the concepts...

Generating Images from Natural Language

Prompt: “A cat walking a dog at University of Michigan central campus”



Generated by [DALL·E 3](#): able to generate **new data**

Introduction

Big data/ deep learning is revolutionizing so many fields...

Days before OpenAI



Days after OpenAI



A great tool (used properly) for software engineer/data scientist/education
(The ethics guideline: <https://genai.umich.edu/guidance/students>)



[Isaac Gym](#) from Nvidia Isaac Sim

Robotics

Grading

Final grades will be based on

Assessment	Percentage
Homework (8)	40%
Midterm	30%
Final Project	30%

An **in-person** midterm is scheduled on :

Wed, 10/08/24

Consider it as a mini-interview for basic Python concepts...

Final Projects (30%)

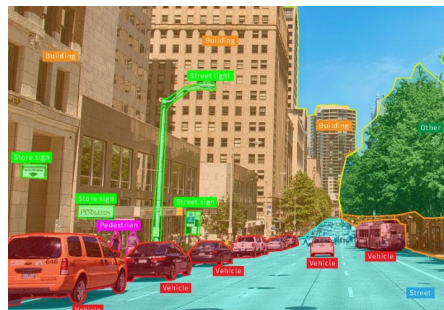
Assuming knowing Python and related tools, the final project for this course is to build ANY end-to-end project with [hugging face](#) where you can use any open-sourced dataset and model (pretrained or fine-tuned).

More details on submission guidelines and evaluation criteria will be provided with the project announcement.

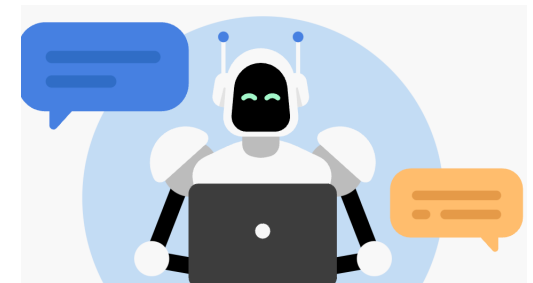
Music generation



Computer Vision



Large Language Models



A general guideline

Time Limit: You have 80 mins to complete this exam.

Total Points: This exam is worth **100** points total.

Question Format: The exam is closed-book and will consist of various question types, including multiple choice, short answers, and writing out the code outputs...

Clarity: Write legibly and explain your reasoning clearly. If we can't read it, we can't grade it.

Tips for preparation: ask what, why and how!

So far...

Write **correct** Python

- Use built-in objects
- Define our own class

Write **effective** Python (evaluate Python)

NumPy

What is Python?

Python is a **dynamically typed, interpreted** programming language

- Created by Guido van Rossum and first released in 1991.
- Design philosophy **simple, readable**

Dynamically typed

In many languages, when you declare a variable, you **must specify the variable's type** (e.g., int, double, Boolean, string). Python does not require this, the type of a variable is **defined at runtime**.

v.s. statically typed, flexible yet more error-prone

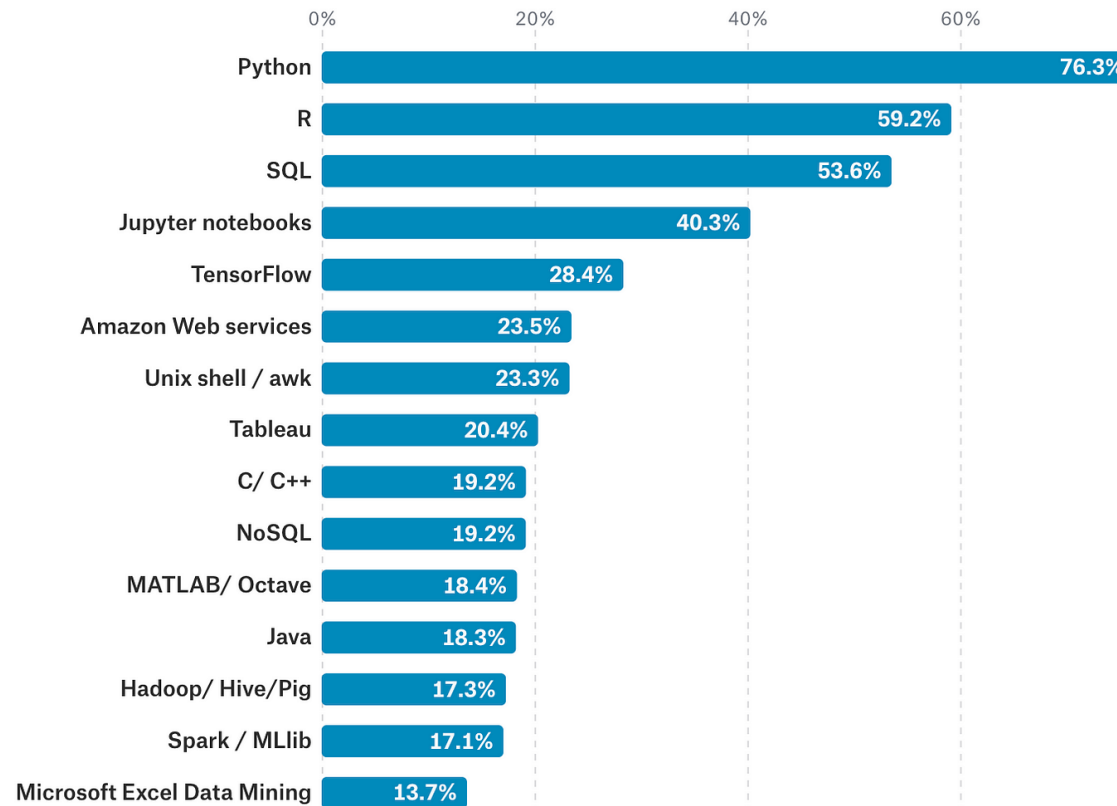
Interpreted

Some languages (e.g. C/C++ and Java) are compiled: we write code, from which we get a runnable program via **compilation**. In contrast, Python is **interpreted**: a program, called the **interpreter**, runs our code directly, line by line.

v.s compiled: simple yet slower

Why Python?

Increasing, Python is the language of data science and general programming.



- Popular
- Easy to learn
- Open-sourced (no license needed)

Data science community Kaggle's annual survey "The State of Data Science& Machine Learning" asks the question "What tools are used at work?" ([reference](#))

Object data types in Python

Different object can **represent** different concepts.

ANY object has a **type** that defines what kind of **operations** programs can do to them

Text Type: `str`

Numeric Types: `int`, `float`, `complex`

Sequence Types: `list`, `tuple`, `range`

Mapping Type: `dict`

Set Types: `set`, `frozenset`

Boolean Type: `bool`

Binary Types: `bytes`, `bytearray`, `memoryview`

None Type: `NoneType`

Ref:

https://www.w3schools.com/python/python_datatypes.asp

String and its operations

String is an **immutable** **sequence** of **case sensitive** characters.

- Letters, special characters, spaces, digits
- “me”, ‘States 507’
- Another built-in **data type** in Python

Create a string (single or double quote)

- str1 = “This is a string”
- str2 = ‘This is also be a string’

Manipulate a string

- **Sequence indexing** (how is it defined and done, potential error message...)
- **Slicing (Syntax)**
- Properties: **immutability**
- **Other string methods and dot notation**

Same for list, tuple and dictionaries

Lists are (**mutable**) sequences whose values can be of **any data type**

- We call those list entries the **elements** of the list

Mutable: We can **change** values of specific elements of a list.

Add new element, **delete** existing ones, **reorder(sort)** and many more...

Tuples are **immutable sequences** whose values can be of **any data type**

Dictionaries are ...

Tips: think about what, how to **create, operations, comparisons**
between different data types...

Recap: Class as programmer-defined types

Objects are instances of a class and are a data abstraction that captures:

- An **internal representation**
 - Through data attributes
- An **interface** for interacting with objects
 - Though methods (aka procedures/functions)
 - Defines behaviors but hides implementations

Creating the class (a parallel to function)

- Define the class **name**
- Define class data attributes
- Define procedural attributes

Using the class:

- Create **new instances** of the class
- Doing operations on the instances

Recap: Inheritance

Inheritance is perhaps the most useful feature of object-oriented programming

Inheritance allows us to create new classes from old ones

Parent class

(base class/superclass)

Class definition Class name

Child class

(derived class/ subclass)

- **Inherit** all data and behaviors of the parent class
- **Add** more info(data)
- **Add** more behavior
- **Override** behavior

Parent class

```
class Cat(Animal):  
    def speak(self):  
        print("meow")
```

```
my_cat = Cat(7)  
my_cat.set_name("Fay")  
print(my_cat.get_name())  
print(my_cat.speak())
```

Fay
meow
None

Recap: Iterator as an object

An iterator is an object that represents a “data stream”

Supports method `__next__()`:

returns next element of the stream/sequence

raises `StopIteration` error when there are no more elements left

```
1 class Squares():
2     '''Iterator over the squares.'''
3     def __init__(self):
4         self.n = 0
5     def __next__(self):
6         (self.n, k) = (self.n+1, self.n)
7         return(k*k)
8     def __iter__(self):
9         return(self)
10 s = Squares()
11 for x in s:
12     print(x)
```

Iterable means that an object has the `__iter__()` method, which returns an iterator. So `__iter__()` returns a new object that supports `__next__()`.

`__next__()` is the important point, here.
It returns a value, the next square.

Now `Squares` supports `__iter__()` (it just returns itself!), so Python allows us to iterate over it.

Recap: Handling exceptions

Exception handler in Python

```
try:                                if <all potentially problematic code succeeds>:
    # do some potentially          # great, all that code
    # problematic code            # just ran fine!
except:                             else:
    # do something to             # do something to
    # handle the problem          # handle the problem
```

Besides `except` blocks

- `else`
 - Body will always be executed **when** `try` block completes with no exceptions
- `finally`
 - Useful for cleanup actions

```
def divide_numbers(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
    else:
        return f"The result is: {result}"
    finally:
        print("Execution complete.")
```

Execution complete.

'The result is: 2.0'

Problems with timing and counting

Timing the exact running time of the program:

- Depends on machine
- Depends in implementation
- Small inputs don't show growth

Counting the exact number of steps:

- Gives a formula
- Do NOT depend on machine
- Depends on the implementation
- Also consider irrelevant operations for largest inputs
 - Initial assignment, addition

Goal:

- Evaluate algorithms (not implementation)
- Evaluate scalability just in terms of input size

Asymptotic growth: the order of growth

We can evaluate programs by

- Timer
- Count the operations
- Abstract notion of order of growth

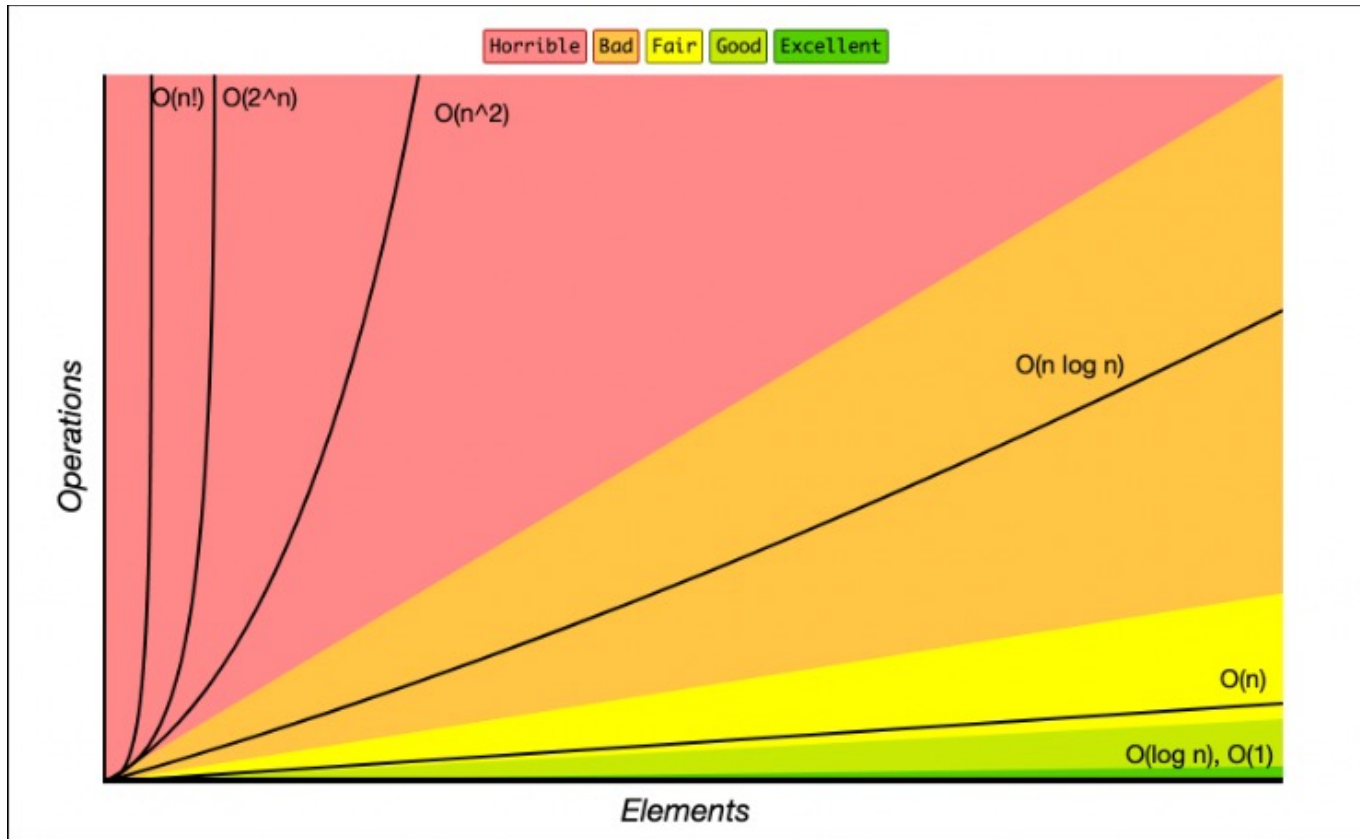
Goal: Describe how run time grows as size of input grows

- Want to put a bound on growth
- Do **NOT** need to be precise: “order of ” not “exact” growth
- Want to focus on terms that grows most rapidly
 - Ignore additive and multiplicative constants

This is called order of growth

- Use mathematical notions of “Big Oh(O)” and “Big Theta(Θ)”

$\Theta(x)$ Complexity Classes



- $\Theta(1)$: denotes **constant** running time
- $\Theta(\log n)$: denotes **logarithmic** running time
- $\Theta(n)$: denotes **linear** running time
- $\Theta(n \log n)$: denotes **log-linear** running time
- $\Theta(n^c)$: denotes **polynomial** running time
- $\Theta(c^n)$: denotes **exponential** running time
- $\Theta(n!)$: denotes **factorial** running time

What's NumPy

Open-sourced add-on modules for **numerical computing**

- 1) NumPy: **numerical** python, have multidimensional arrays
- 2) Optimized library for **matric and vector computation**
- 3) Makes uses of C/C++ subroutines and memory-efficient data structure
- 4) Building block for other packages: SciPy, Matplotlib, scikit-learn, scikit-image and provides fast numerical computations and high-level math functions

Why NumPy (v.s. built-in lists)

Python is very slow and NumPy are much more **efficient**

- 1000 x 1000 matrix multiply
 - Python triple loop takes > 10 min.
 - NumPy takes ~0.03 seconds

Have more advanced mathematical functions, **convenient**

- Have mathematical operations applied directly to arrays
 - Linear algebra, statistical operations...

Broadcasting and vectorization saves time and amount of code

Example questions

1. Multiple Choice Questions. You must identify all correct options to receive full points for the question. Partial credit will not be awarded for partially correct responses.

(a) (3 points) What will be the output of the following code?

```
1 numbers = [5, 10, 15, 20, 10]
2 numbers.append(25)
3 numbers.remove(10)
4 result = numbers.pop()
5 print(numbers)
```

A

- ☐ [5, 15, 20, 10]
- ☐ [5, 15, 20]
- ☐ [10, 15, 20, 10]
- ☐ [5, 10, 15, 20]

(b) (3 points) Which of the following correctly creates a tuple containing only the integer 1?

- ☐ t = (1)
- ☐ t = (1,)
- ☐ t = tuple("1")
- ☐ t = tuple([1])

B,D

Example questions

2. Please answer these questions with one or two complete sentences.

(a) (4 points) In Python, can you change the type of a variable after it's been declared? If so, demonstrate with a short code snippet.

(b) (4 points) What is the purpose of the `__init__` method in Python classes?

Example questions

3. For each of the following code snippets, identify what type of error will occur (2 points) and explain why (1 point).

(a) (3 points) List, Dictionary and String Manipulations 1.

```
1 student_scores = {"Alice": 95, "Bob": 87, "Charlie": 92}
2 del student_scores["David"]
3 print(student_scores)
```

KeyError

Key does not exist in the dictionary

4. For each question, write the expected output of the given code snippet. If there's an error, explain what kind of error would occur.

(a) (3 points) Error Handling 1

```
1 try:
2     y = 10 / 0
3 except ZeroDivisionError:
4     print("Division by zero!")
5 else:
6     print("else block")
7 finally:
8     print("finally block")
```

Division by zero!

finally block

Example questions

5. Time Complexity Analysis. For each of the following code snippets, determine the time complexity in Big θ notation.

(a) (4 points) Nested Loop Analysis.

```
1 def nested_operations(n):
2     count = 0
3     for i in range(n):
4         for j in range(n):
5             count += 1
6             if count % 2 == 0:
7                 for k in range(10): # Fixed iteration of 10
8                     count += 1
9     return count
```

$\theta(n^2)$

Good luck!

Questions?