Lecture 7
Object Oriented Programming

Jan Tekülve
jan.tekuelve@ini.rub.de

Computer Science and Mathematics Preparatory Course

02.10.2019
Overview

1. Object Oriented Programming
   - What is OOP?
   - Example Project
   - Inheritance
   - Modules in Python

2. Tasks

3. Outlook: Matrices and Scientific Programming
   - Matrices Quick Summary
   - The Numpy Module
   - Matrix Calculation with Numpy
Programming Paradigms

Procedural Programming

- A problem is solved by manipulating data structures through procedures
- The key is to write the right logic
- Efficiency is a main focus of procedural programming
### Programming Paradigms

**Procedural Programming**

- A problem is solved by manipulating data structures through procedures
- The key is to write the right logic
- Efficiency is a main focus of procedural programming

**Object oriented Programming**

- A problem is solved by modeling its processes
- The key is to figure out the relevant entities and their relations
- Programming Logic is tightly coupled to entities
Classes vs. Objects

Class

Person
first name
last name
age
email
Classes vs. Objects

**Class**

Person

- first name
- last name
- age
- email

**Objects (Instances)**

- Alice Anderson
  - age: 28
  - email: a.anders@gmail.com

- Rob Robertson
  - age: 17
  - email: cool_dude@aol.com
Classes Bind Variables Together

- Instead of writing something like this

```python
# Alice's attributes
alice_name = "Alice"
alice_last_name = "Anderson"
alice_age = 28
```
Classes Bind Variables Together

- Instead of writing something like this

```python
#Alice's attributes
define alice_name = "Alice"
define alice_last_name = "Anderson"
define alice_age = 28
```

- Objects encapsulate multiple variables in one place

```python
#A Person-object variable
define alice = Person("Alice","Anderson",28)
```
Classes are Advanced Data Types

▶ Object variables can be treated like simple types

```python
#Two Person-object variables
alice = Person("Alice","Anderson",28)
rob = Person("Rob","Robertson",17)
#Objects can be stored in lists
myPersonList = [] #I want to manage persons
myPersonList.append(rob)
#Objects can be arguments of self-defined functions
calculate_year_of_birth(alice)
```
Class Definition

A class needs to be defined

class Person: #This defines the class Name
    #The __init__ function is responsible for class creation
    #and defines its' attributes
    def __init__(self, first_name, last_name, age):
        #The passed values are stored in the class
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

This is enough to create a class-object

robbie = Person("Rob","Robertson",17)
Accessing Class Attributes

Class attributes can be accessed via the ‘.’ operator

```python
robb = Person("Rob","Robertson",17)

f_name = robb.first_name #"Rob"
l_name = robb.last_name #"Robertson"
age = robb.age #17
```
Accessing Class Attributes

- Class attributes can be accessed via the ‘.’ operator

```python
robbie = Person("Rob","Robertson",17)

f_name = robbie.first_name #"Rob"
l_name = robbie.last_name #"Robertson"
age = robbie.age #17
```

- They can also be assigned after initialization

```python
robbie.age = 18 #As he gets older
robbie.l_name = "Peterson" #If he marries
```
Objects and Functions

We can use objects as function arguments

#Definition

def print_info(person):
    print(person.first_name + " " + person.last_name + " is " + str(person.age) + " years old.")
**Objects and Functions**

- We can use objects as function arguments

```python
# Definition
def print_info(person):
    print(person.first_name +" " +person.last_name +" is " +str(person.age) +" years old.")

# Usage:
robby = Person("Rob","Robertson",17)
print_info(robby)
# This prints: "Rob Robertson is 17 years old"

alice = Person("Alice","Anderson",28)
print_info(alice)
# This prints: "Alice Anderson is 28 years old"
```
Function Encapsulation

Functions can even be defined inside classes

class Person: #This defines the class Name
    #The __init__ function
    def __init__(self, first_name, last_name, age):
        #The passed values are stored in the class
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    #Our print_info function
    def print_info(self):
        print(self.first_name + " " + str(self.age) + " years old.")
**Function Encapsulation**

- A function can be called directly from the object

```python
robbie = Person("Rob","Robertson",17)
robbie.print_info()
#This prints: "Rob Robertson is 17 years old"

alice = Person("Alice","Anderson",28)
alice.print_info()
#This prints: "Alice Anderson is 28 years old"
```
Function Encapsulation

▶ A function can be called directly from the object

```python
robbie = Person("Rob","Robertson",17)
robbie.print_info()
#This prints: "Rob Robertson is 17 years old"

alice = Person("Alice","Anderson",28)
alice.print_info()
#This prints: "Alice Anderson is 28 years old"
```

▶ This way a potential programmer/user does not need to know the internal structure of the particular class, e.g. `list.append()`.
Course Management Program

- We want to write a program for the university
- It should give an overview over the different courses
- It should track each course, its lecturer and its students
Course Management Program

- We want to write a program for the university
- It should give an overview over the different courses
- It should track each course, its lecturer and its students

How would an OOP model look like?
# Course Management Program

```plaintext
Course
  name
  year
  id_number
  lecturer
  student_list
```
Course Management Program

Course
- name
- year
- id_number
- lecturer
- student_list

Lecturer
- first name
- last name
- age
- email
- bank_account
Course Management Program

Course
- name
- year
- id_number
- lecturer
- student_list

Lecturer
- first name
- last name
- age
- email
- bank_account

Student
- first name
- last name
- age
- email
- student_id
- grade
Example Code

The course class

class Course: #This defines the class Name
    #The __init__ function
def __init__(self, name, year, id_number, lecturer):
    #The passed values are stored in the class
    self.name = name
    self.year = year
    self.id_number = id_number
    self.lecturer = lecturer
    self.student_list = [] #empty upon creation
Example Code

The lecturer class

class Lecturer: #This defines the class Name
    #The __init__ function
    def __init__(self, first_name, last_name, age, email,
                  bank_account):
        #The passed values are stored in the class
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.email = email
        self.bank_account = bank_account
Example Code

Create the Course

```python
lecturer_jan = Lecturer("Jan","Tekuelve",30,"jan.tekuelve@ini.rub.de",1234567)
cscience_course = Course("Computer Science and Mathematics",2019,1234,lecturer_jan)
```

At the end of the year access the bank account:
```
c_bank_account = cscience_course.lecturer.bank_account
```

/T_h.ligais works independent of course and lecturer

```text
/zero.lf/two.lf/one.lf/zero.lf/two.lf/one.lf/nine.lf/one.lf/two.lf/three.lf/five.lf
```
Example Code

Create the Course

```python
lecturer_jan = Lecturer("Jan","Tekuelle",30,"jan.tekuelle@ini.rub.de",1234567)
cscience_course = Course("Computer Science and Mathematics",2019,1234,lecturer_jan)
```

At the end of the year access the bank account:

```python
c_bank_account = cscience_course.lecturer.bank_account
```
Example Code

Create the Course

```python
lecturer_jan = Lecturer("Jan","Tekuelve",30,"jan.tekuelve@ini.rub.de",1234567)
cscience_course = Course("Computer Science and Mathematics",2019,1234,lecturer_jan)
```

At the end of the year access the bank account:

```python
c_bank_account = cscience_course.lecturer.bank_account
```

This works independent of course and lecturer
The Student Class

This class looks similar to the lecturer

```python
class Student:  #This defines the class Name
    #The __init__ function
    def __init__(self, first_name, last_name, age, email, student_id):
        #The passed values are stored in the class
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.email = email
        self.student_id = student_id
        self.grade = -1
```
Code Redundancy

Course
- name
- year
- id_number
- lecturer
- student_list

Lecturer
- first name
- last name
- age
- email
- bank_account

Student
- first name
- last name
- age
- email
- student_id
- grade
Code Redundancy

Course
name
year
id_number
lecturer
student_list

Lecturer
first name
last name
age
e-mail
bank_account

Student
first name
last name
age
e-mail
student_id
grade
Code Redundancy

Course
- name
- year
- id_number
- lecturer
- student_list

Person
- first name
- last name
- age
- email

Lecturer
- first name
- last name
- age
- email
- bank_account

Student
- first name
- last name
- age
- email
- student_id
- grade
Code Redundancy

Course
- name
- year
- id_number
- lecturer
- student_list

Lecturer
- first name
- last name
- age
- email
- bank_account

Person
- first name
- last name
- age
- email

Student
- first name
- last name
- age
- email
- student_id
- grade
Code Redundancy

Course
- name
- year
- id_number
- lecturer
- student_list

Person
- first name
- last name
- age
- email

Lecturer
- bank_account

Student
- student_id
- grade
The Person Class

We will use the Class Person as Super-Class

class Person: #This defines the class Name
    #The __init__ function
    def __init__(self, first_name, last_name, age, email):
        #The passed values are stored in the class
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.email = email
Inheritance

Lecturer and Student will inherit from Person

class Lecturer(Person): #Brackets declare inheritance
    #The __init__ function is overridden
    def __init__(self,f_name,l_name,age,email,b_acc):
        #The super() calls the parent function
        super().__init__(f_name,l_name,age,email)
        self.bank_account = b_acc

class Student(Person): #Brackets declare inheritance
    #The __init__ function is overridden
    def __init__(self,f_name,l_name,age,email,stud_id):
        super().__init__(f_name,l_name,age,email)
        self.student_id = stud_id
        self.grade = -1
Modifying the Parent Class

Functions of the parent class are available to child classes

```python
class Person: #This defines the class Name
    def __init__(self, first_name, last_name, age, email):
        #The passed values are stored in the class
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.email = email

    #Our print_info function
    def print_info(self): #Note how the argument changed
        print(self.first_name + " " + self.last_name + " is" +str(self.age) +" years old.")
```
Using Parent Functions

Functions of the parent class are available to child classes

```python
student_rob = Student("Rob","Robertson",25,"rob.
    ↞ robson@rub.de","108001024")
lecturer_jan = Lecturer("Jan","Tekuelve",30,"jan.
    ↞ tekuelve@ini.rub.de",1234567)

student_rob.print_info()
lecturer_jan.print_info()
#Prints:
#Rob Robertson is 25 years old.
#Jan Tekuelve is 30 years old.
```
Completing the Example

- The course needs to be able to add students

  #Inside the Course class
def enroll(self, student):
      self.student_list.append(student)
  #Enroll adds them to the course internal list

- Minimal example:
cscience_course = Course("Computer Science and Mathematics", 2019, 1234, lecturer_jan)
student_rob = Student("Rob", "Robertson", 25, "rob.robson@rub.de", "108001024")
cscience_course.enroll(student_rob)
Creating your own Python Modules

- Class definitions can be stored in separate module

- E.g. if you save the above class definitions in a file `unimanager.py`
Creating your own Python Modules

- Class definitions can be stored in separate module
- E.g. if you save the above class definitions in a file `unimanager.py`
- You can access the definitions in another script from the same folder:

```python
import unimanager
student_rob = unimanager.Student("Rob","Robertson",25,"rob.robson@rub.de","108001024")
```
Creating your own Python Modules

- Class definitions can be stored in separate module
- E.g. if you save the above class definitions in a file `unimanager.py`
- You can access the definitions in another script from the same folder:
  ```python
  import unimanager
  student_rob = unimanager.Student("Rob","Robertson",25,"rob.robson@rub.de","108001024")
  ```
- This allows for flexible re-usability of code
Advantages/Disadvantages of OOP

Advantages:

▶ Design Benefit: Real/World processes are easily transferable in code
▶ Modularity: Extending and reusing software is easy
▶ Software Maintenance: Modular code is easier to debug

Disadvantages:

▶ Design Overhead: Modeling requires longer initial development time
▶ Originally OOP required more "coding"
Advantages/Disadvantages of OOP

**Advantages:**

- Design Benefit: Real/World processes are easily transferable in code
- Modularity: Extending and reusing software is easy
- Software Maintenance: Modular code is easier to debug

**Disadvantages:**

- Design Overhead: Modeling requires longer initial development time
- Originally OOP required more “coding”
Tasks

1. Download today's class definitions `unimanager.py` and create a separate script that uses this module to create a course, a lecturer and three sample students.
   - Enroll all students to the course.
   - After enrolling iterate through the student list to print the info of all enrolled students. You can access the student_list via the course object.
   - In the loop use the `print_info()` function.

2. Add a `print_info()` function to the class definition of Course in `unimanager.py`. This function should print the course name, its lecturer and each student of the course with his/her student ID.
   - The function should be defined in the Course class and its only argument should be self
   - The course name, the lecturer and its student_list can be accessed via the self keyword.
Matrix Definition

A Matrix $A_{m,n}$ is a rectangular array arranged in $m$ rows and $n$ columns.

Example:

$A_{3,4} = \begin{pmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{pmatrix}$
Matrix Definition

A Matrix $A_{m,n}$ is a rectangular array arranged in $m$ rows and $n$ columns.

Example:

$$A_{3,4} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

A single element in a matrix is usually denoted by $a_{i,j}$, where $i$ is the row and $j$ the column index. For example $a_{2,3} = 7$. 
Matrix Definition

A Matrix $A_{m,n}$ is a rectangular array arranged in $m$ rows and $n$ columns.

- Example:

$$A_{3,4} = \begin{pmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{pmatrix}$$

- A single element in a matrix is usually denoted by $a_{i,j}$, where $i$ is the row and $j$ the column index. For example $a_{2,3} = 7$.

- A matrix $A_{m, n}$, where $m = n$ is called a square matrix
Matrix Definition

A Matrix $A_{m,n}$ is a rectangular array arranged in $m$ rows and $n$ columns.

- Example:

$$A_{3,4} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

- A single element in a matrix is usually denoted by $a_{i,j}$, where $i$ is the row and $j$ the column index. For example $a_{2,3} = 7$.

- A matrix $A_{m,n}$, where $m = n$ is called a square matrix.

- A matrix that has only entries on the diagonal is called a diagonal matrix.

$$D_{3,3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{Special case identity matrix} \quad I_{3,3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
Matrix Addition/Subtraction

It is possible to add two matrices $A$ and $B$ together, if they have the same number of rows and columns.
Matrix Addition/Subtraction

- It is possible to add two matrices $A$ and $B$ together, if they have the same number of rows and columns.

- Addition is carried out element-wise:

$$A_{3,2} + B_{3,2} = \begin{pmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \end{pmatrix} + \begin{pmatrix} 4 & 2 \\ 3 & 1 \\ 8 & 2 \end{pmatrix} = \begin{pmatrix} 1+4 & 2+2 \\ 5+3 & 6+1 \\ 9+8 & 10+2 \end{pmatrix} = \begin{pmatrix} 5 & 4 \\ 8 & 7 \\ 17 & 12 \end{pmatrix}$$
Matrix Addition/Subtraction

- It is possible to add two matrices $A$ and $B$ together, if they have the same number of rows and columns.

- Addition is carried out element-wise:

\[
A_{3,2} + B_{3,2} = \begin{pmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \end{pmatrix} + \begin{pmatrix} 4 & 2 \\ 3 & 1 \\ 8 & 2 \end{pmatrix} = \begin{pmatrix} 1+4 & 2+2 \\ 5+3 & 6+1 \\ 9+8 & 10+2 \end{pmatrix} = \begin{pmatrix} 5 & 4 \\ 8 & 7 \\ 17 & 12 \end{pmatrix}
\]

- Subtraction works analogously:

\[
A_{3,2} - B_{3,2} = \begin{pmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \end{pmatrix} - \begin{pmatrix} 4 & 2 \\ 3 & 1 \\ 8 & 2 \end{pmatrix} = \begin{pmatrix} 1-4 & 2-2 \\ 5-3 & 6-1 \\ 9-8 & 10-2 \end{pmatrix} = \begin{pmatrix} -3 & 0 \\ 2 & 5 \\ 1 & 8 \end{pmatrix}
\]
Scalar Multiplication and Transposition

Multiplication with scalar values is also applied element-wise:

\[
A_{3,2} \cdot 3 = \begin{pmatrix}
1 & 2 \\
5 & 6 \\
9 & 10
\end{pmatrix} \cdot 3 = \begin{pmatrix}
1 \cdot 3 & 2 \cdot 3 \\
5 \cdot 3 & 6 \cdot 3 \\
9 \cdot 3 & 10 \cdot 3
\end{pmatrix} = \begin{pmatrix}
3 & 6 \\
15 & 18 \\
27 & 30
\end{pmatrix}
\]
Scalar Multiplication and Transposition

- Multiplication with scalar values is also applied element-wise:

\[
A_{3,2} \cdot 3 = \begin{pmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \end{pmatrix} \cdot 3 = \begin{pmatrix} 1 \cdot 3 & 2 \cdot 3 \\ 5 \cdot 3 & 6 \cdot 3 \\ 9 \cdot 3 & 10 \cdot 3 \end{pmatrix} = \begin{pmatrix} 3 & 6 \\ 15 & 18 \\ 27 & 30 \end{pmatrix}
\]

- The transposition \(A^T\) of a matrix switches the roles of row and columns.

Example:

\[
A_{3,2}^T = \begin{pmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \end{pmatrix}^T = \begin{pmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \end{pmatrix}
\]

The transposition turns a \(m \times n\) matrix into a \(n \times m\) matrix.
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$. 
Matrix Multiplication

Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m \times n}$ matches the number of rows in $B_{n \times o}$.

The resulting matrix $C_{m \times o}$ shares the number of rows from $A$ and the number of columns from $B$.

Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

**Multiply Row by Column**

$$A_{2,3} \cdot B_{3,3} = \begin{pmatrix} 3 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 8 \\ 1 & 2 & 10 \\ 7 & 3 & 2 \end{pmatrix} = \begin{pmatrix} \_ & \_ & \_ \\ \_ & \_ & \_ \\ \_ & \_ & \_ \end{pmatrix}$$
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$.

- Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

**Multiply Row by Column**

$$A_{2,3} \cdot B_{3,3} = \begin{pmatrix} 3 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 8 \\ 1 & 2 & 10 \\ 7 & 3 & 2 \end{pmatrix} = \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \end{pmatrix}$$
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$.

- Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

**Multiply Row by Column**

$$A_{2,3} \cdot B_{3,3} = \begin{pmatrix} 3 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 8 \\ 1 & 2 & 10 \\ 7 & 3 & 2 \end{pmatrix} = \begin{pmatrix} (3 \cdot 4 + 6 \cdot 1 + 5 \cdot 7) & - & - \\ - & - & - \end{pmatrix}$$
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$.

- Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

**Multiply Row by Column**

$$A_{2,3} \cdot B_{3,3} = \begin{pmatrix} 3 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 8 \\ 1 & 2 & 10 \\ 7 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 53 & - & - \\ - & - & - \end{pmatrix}$$
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$.

- Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

**Multiply Row by Column**

$$A_{2,3} \cdot B_{3,3} = \begin{pmatrix} 3 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 8 \\ 1 & 2 & 10 \\ 7 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 53 & ? & _- \\ _- & _- & _- \end{pmatrix}$$
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$.

- Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

**Multiply Row by Column**

$$
A_{2,3} \cdot B_{3,3} = \begin{pmatrix}
3 & 6 & 5 \\
4 & 2 & 1
\end{pmatrix} \cdot 
\begin{pmatrix}
4 & 3 & 8 \\
1 & 2 & 10 \\
7 & 3 & 2
\end{pmatrix} = \begin{pmatrix}
53 & (3 \cdot 3 + 6 \cdot 2 + 5 \cdot 3) & - \\
- & - & -
\end{pmatrix}
$$
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$.

- Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

Multiply Row by Column

$$A_{2,3} \cdot B_{3,3} = \begin{pmatrix} 3 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 8 \\ 1 & 2 & 10 \\ 7 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 53 & 36 & \_ \\ \_ & \_ & \_ \end{pmatrix}$$
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$.

- Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

**Multiply Row by Column**

\[
A_{2,3} \cdot B_{3,3} = \begin{pmatrix} 3 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 8 \\ 1 & 2 & 10 \\ 7 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 53 & 36 & - \\ - & - & ? \end{pmatrix}
\]
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$.

- Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

**Multiply Row by Column**

$$A_{2,3} \cdot B_{3,3} = \begin{pmatrix} 3 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 8 \\ 1 & 2 & 10 \\ 7 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 53 & 36 & - \\ - & - & (4 \cdot 8 + 2 \cdot 10 + 1 \cdot 2) \end{pmatrix}$$
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$.

- Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

Multiply Row by Column

$$A_{2,3} \cdot B_{3,3} = \begin{pmatrix} 3 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 8 \\ 1 & 2 & 10 \\ 7 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 53 & 36 & - \\ - & - & 54 \end{pmatrix}$$
Matrix Multiplication

- Matrices $A$ and $B$ can be multiplied with each other, if the number of columns of $A_{m,n}$ matches the number of rows in $B_{n,o}$.

- The resulting matrix $C_{m,o}$ shares the number of rows from $A$ and the number of columns from $B$.

- Matrix multiplication is carried out by multiplying the row-vector of the first matrix with the column-vector of the second matrix.

**Multiply Row by Column**

$$A_{2,3} \cdot B_{3,3} = \begin{pmatrix} 3 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 8 \\ 1 & 2 & 10 \\ 7 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 53 & 36 & 94 \\ 25 & 19 & 54 \end{pmatrix}$$
The Numpy Module

- Numpy is part of SciPy the module for scientific programming
- It should have been installed with matplotlib
- It is usually imported like this:

```python
import numpy as np
```
The Numpy Array

Numpy brings its own data structure the numpy array

```python
import numpy as np
#Arrays can be created from lists
array_example = np.array([1, 6, 7, 9])
#Arrays can be created with arange
#An array with numbers from 4 to 5 and step size 0.2
array2 = np.arange(4, 5, 0.2)  # 5 is not in the array
print(array2)  # [4.0 4.2 4.4 4.6 4.8]
```

Elements of an array can be manipulated simultaneously

```python
array3 = array2*array2  # For example with multiplication
print(array3)  # [16.0 16.64 19.36 21.16 23.04]
```
Matplotlib and Numpy

Plotting $\sin(x)$ from 0 to $\pi$ with lists

```python
listX=[]
listY=[]
step_size = 0.5
for i in range(0,int(math.pi/step_size)):
    xValue = i*step_size
    listX.append(xValue)
    listY.append(math.sin(xValue))
plt.plot(listX,listY)
```

Plotting $\sin(x)$ from 0 to $\pi$ with numpy

```python
xValues = np.arange(0,math.pi,0.5)
yValues = np.sin(xValues)
plt.plot(xValues,yValues)
```
Numpy Arrays as Matrices

Creating the following matrix: \( A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \)
Numpy Arrays as Matrices

▶ Creating the following matrix: \( \mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \)

▶ In numpy a matrix can be created from a multi-dimensional list

# This creates a 3x4 Matrix

\[
A = \text{np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])}
\]
Numpy Arrays as Matrices

- Creating the following matrix: 
  \[
  A = \begin{pmatrix}
  1 & 2 & 3 & 4 \\
  5 & 6 & 7 & 8 \\
  9 & 10 & 11 & 12 \\
  \end{pmatrix}
  \]

- In numpy a matrix can be created from a multi-dimensional list

  ```python
  # This creates a 3x4 Matrix
  A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
  ```

- Numpy treats such an array as a matrix

  ```python
  arr_dim = A.shape  # Gives you the shape of your matrix
  print(arr_dim)  # Prints (3,4)
  # Access elements with indexing
  single_number = A[1,3]  # 8, 2nd list, 4th element
  num2 = A[0,1]  # 2, 1st list, 2nd element
  ```
Matrix Operations in Numpy

Matrix Addition: \[
\begin{pmatrix}
1 & 2 & 3 \\
5 & 6 & 7 \\
\end{pmatrix}
+ 
\begin{pmatrix}
3 & 5 & 1 \\
5 & -3 & 1 \\
\end{pmatrix}
= 
\begin{pmatrix}
4 & 7 & 4 \\
10 & 3 & 8 \\
\end{pmatrix}
\]

In numpy code:

```python
A = np.array([[1, 2, 3], [5, 6, 7]])
B = np.array([[3, 5, 1], [5, -3, 1]])
C = A + B
D = A - B  # Subtraction works analogously
print(D)  # [[-2 -3  2], [0  9  6]]
```
Matrix Operations in Numpy

- **Matrix Multiplication:**
  \[
  \begin{pmatrix}
  1 & 2 & 3 \\
  5 & 6 & 7 \\
  \end{pmatrix}
  \times
  \begin{pmatrix}
  3 & 5 \\
  5 & -3 \\
  1 & 1 \\
  \end{pmatrix}
  =
  \begin{pmatrix}
  16 & 2 \\
  52 & 14 \\
  \end{pmatrix}
  \]

- **In numpy code:**
  ```python
  A = np.array([[1,2,3], [5,6,7]])
  E = np.array([[3,5], [5,-3],[1,1]])
  F = np.matmul(A,E)
  print(F) # [[16,2], [52,14]]
  ```

- Do not confuse with element-wise multiplication
  ```python
  A = np.array([[1,2,3], [5,6,7]])
  B = np.array([[3,5,1], [5,-3,1]])
  G = A*B # [[3,10,3], [25,-18,7]]
  ```
Matrix Operations in Numpy

▶ Matrix Multiplication:
\[
\begin{pmatrix}
1 & 2 & 3 \\
5 & 6 & 7
\end{pmatrix}
\times
\begin{pmatrix}
3 & 5 \\
5 & -3 \\
1 & 1
\end{pmatrix}
= \begin{pmatrix}
16 & 2 \\
52 & 14
\end{pmatrix}
\]

▶ In numpy code:
```
A = np.array([[1,2,3], [5,6,7]])
E = np.array([[3,5], [5,-3],[1,1]])
F = np.matmul(A,E)
pdint(F) # [[16,2],[52,14]]
```

▶ Do not confuse with element-wise multiplication
```
A = np.array([[1,2,3], [5,6,7]])
B = np.array([[3,5,1], [5,-3,1]])
G = A*B # [[3,10,3], [25,-18,7]]
```
Matrix Operations in Numpy

- It also works for vectors:

\[ \langle v_1, v_2 \rangle = v_1^T v_2 = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 3 \\ 5 \\ 1 \end{pmatrix} = 16 \]

- In numpy code:

```python
V1 = np.array([1,2,3])
V2 = np.array([3,5,1])
R = np.matmul(V1,V2)
print(R) # 16
```
Matrix Operations in Numpy

► It also works for vectors:

\[
\langle v_1, v_2 \rangle = v_1^T v_2 = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 3 \\ 5 \\ 1 \end{pmatrix} = 16
\]

► In numpy code:

```python
V1 = np.array([1,2,3])
V2 = np.array([3,5,1])
R = np.matmul(V1,V2)
print(R) # 16
```

► Or vectors and matrices if you want to
Other helpful Operations

- Transpose Matrices: \( A = \begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \end{pmatrix} \quad A^T = \begin{pmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \end{pmatrix} \)

- In numpy:

  ```python
  A = np.array([[1,2,3], [5,6,7]])
  H = A.T  # [[1,5],[2,6],[3,7]]
  ```

- Element-wise summing across arrays:

  ```python
  sum = np.sum(H)  # 24,
  V1 = np.array([1,2,3])  # works also for 1D-arrays
  sum_v = np.sum(V1)  # 6
  ```
This concludes the Preparatory Course.

Any Questions or Feedback?