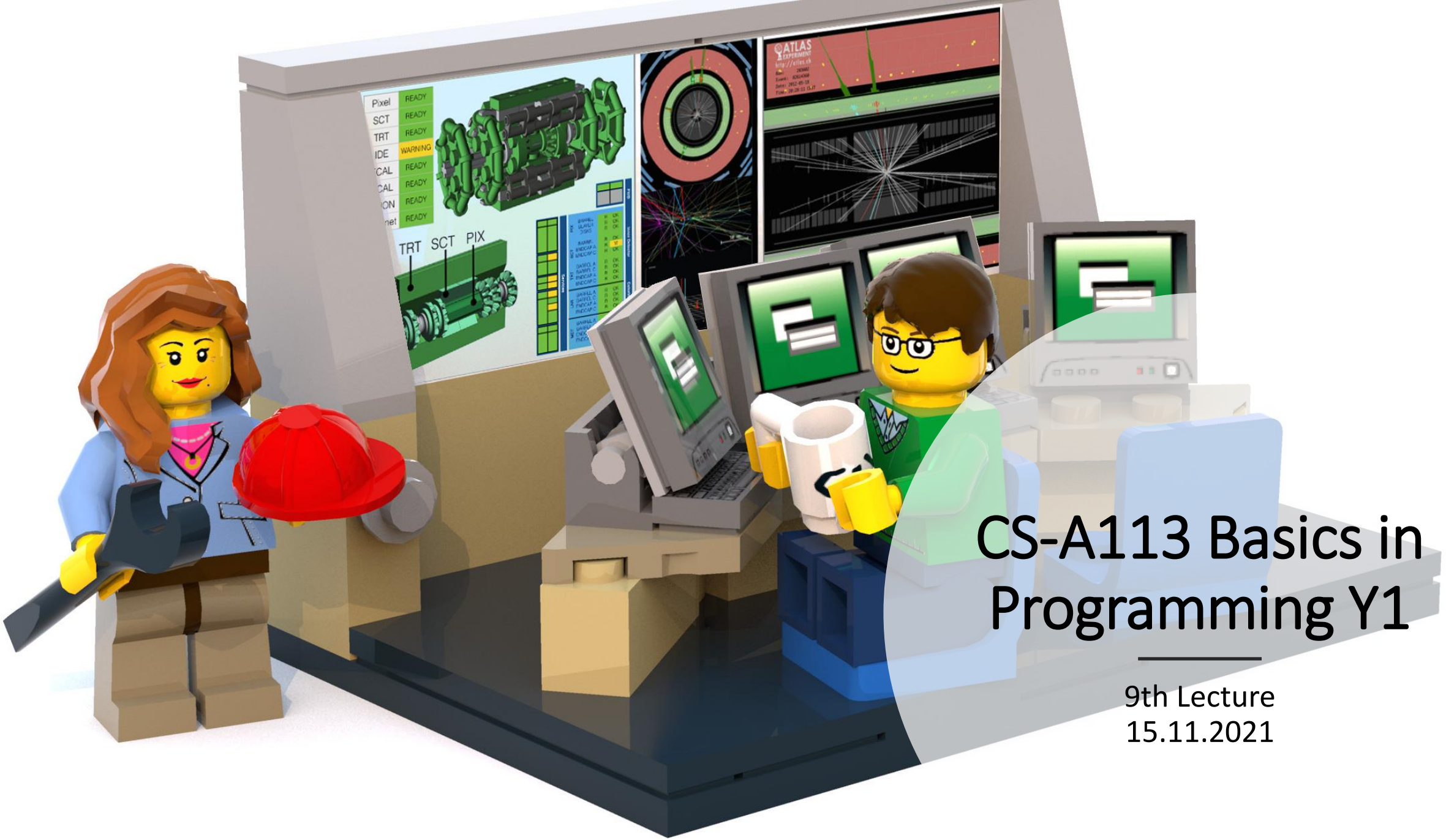


CS-A113 Basics in Programming Y1

9th Lecture
15.11.2021

The Lecture

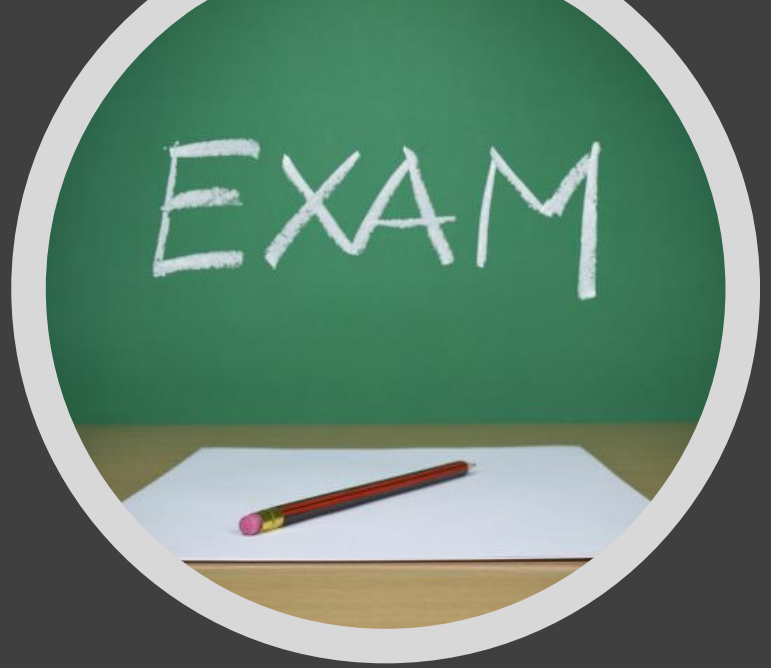
- **Join with Video** – Makes my life nicer!
- Feel free to open your microphone and ask questions
- Feel free to write questions into the chat
- We will record the sessions and put it unlisted on youtube.

Interactions Today:

Course Information

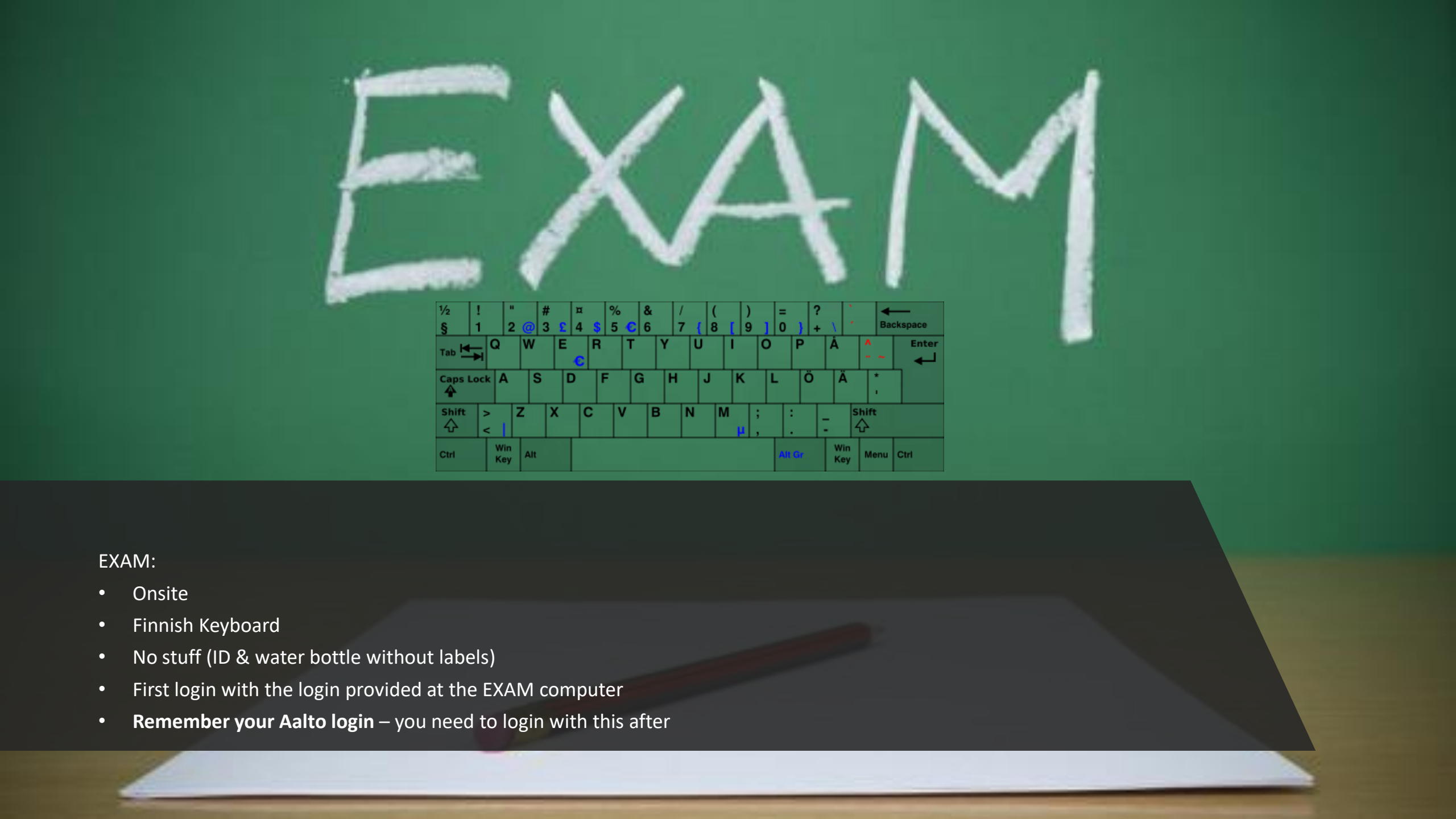
THANK YOU – For your Feedback

Question (# student answers)	Course Start 152	Mid-Course 127
Programming is (bad = 1 / good = 5)	3.5	4.0
How do you feel about this course		
Excited	3.9	3.6
Bored	1.6	1.5
Anxious	2.9	2.6
overwhelmed	2.8	2.9
annoyed	1.6	2.4
Not challenged	1.6	1.7
Agitated	2.0	2.6
happy	3.6	3.6
neutral	2.9	2.7



Course Information

1. Lecture of your choice next week: → **Tell me which topics to revisit until Thursday 18. in the evening at 18:00**
2. **Last exercise sessions on 01.12.**
3. **Q&A session on 30.11 → send topics and questions in advance**
4. Exam takes place as EXAM
    1. Don’t forget to register in SISU for the exam at least 1 week before the exam
    2. Don’t forget to also register for your EXAM slot.
    3. You find information on our myCourses page

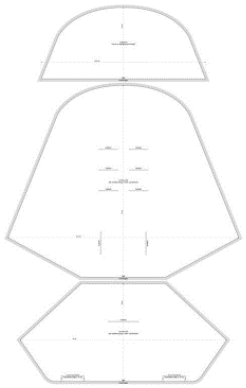


EXAM:

- Onsite
- Finnish Keyboard
- No stuff (ID & water bottle without labels)
- First login with the login provided at the EXAM computer
- **Remember your Aalto login** – you need to login with this after


Topic Today:
OO-Programming

Recap



Sewing Pattern

- You can build more than one object from it
- It describes the underlying structure
- It is not an object itself

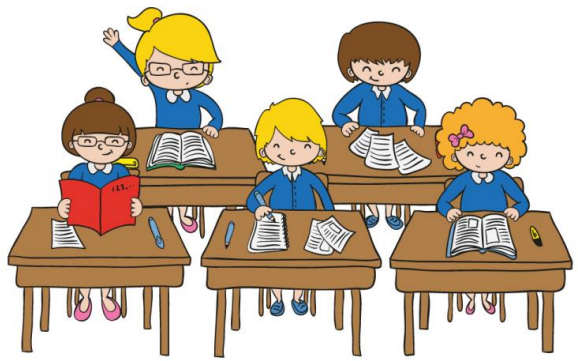
A student has

- a name
- a student number
- courses he/she is enrolled in
- grades

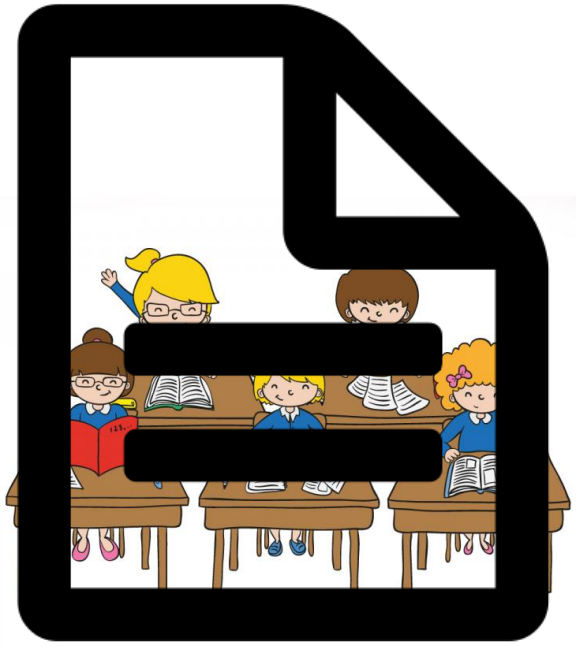
Recap:

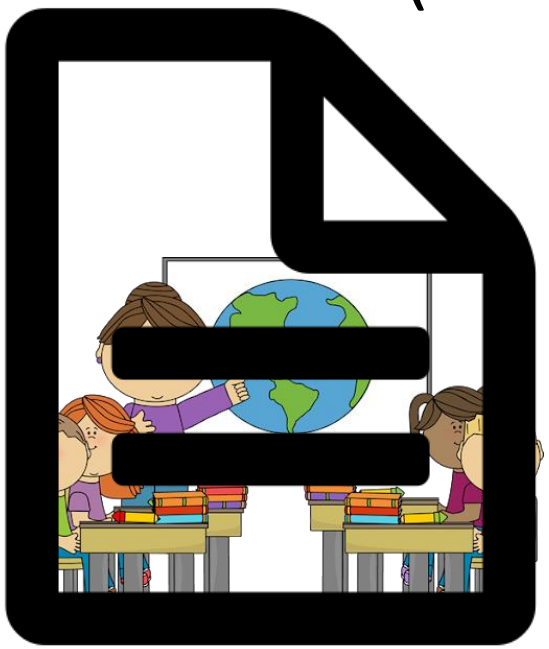
```
class Student:

    def __init__(self, myName, myNumber):
        self.__name = myName
        self.__id = myNumber
        self.__grades = [ ]
        self.__courses = [ ]

    def add_course(self,course):
        self.__courses.append(course)

main():
    student1 = Student(”Barbara”,123)
    student2 = Student(”Angelina”,564)
    studentRegistry = (student1,student2,student3)
    name = read_input()
    student1.add_course(”Basics in Programming”)
    student2.add_course(”Algorithms and Datastructures”)
```

Put Classes to their own File (Module)



Put Classes to their own File (Module)





In our Example:

Class = Student
Module = student
Filename (of the module with the class Student) = student.py

Implement your class once and use it everywhere!

A module is a file containing Python definitions and statements. The file name is the module name with the suffix “.py” appended.

Concept	Example
Module	student
Class	Student
Constructor initializer method	def _ _ init_ _ (self, myName) self._ _name = myName
Initialization	Student1 = Student(“Barbara”, 123)
instance	student1 = Student(“Barbara”, 123) student1
attribute	_ _ name
Method	def add_course(self,course): self._ _courses.append(course)
Method call	student1.add_course(“A”)
Function	def read_input(): return(input(“Enter your input\n”))
Function call	myInput = read_input()


```
class Student:

    def _ _ init_ _(self, myName, myNumber):
        self._ _ name = myName
        self._ _id = myNumber
        self._ _grades = [ ]
        self._ _courses = [ ]

    def add_course(self,course):
        self._ _courses.append(course)

    def sort_courses(self):
        self._ _ courses.sort()

main():
    student1 = Student(”Barbara”,123)
    student2 = Student(”Angelina”,564)
    studentRegistry = (student1,student2,student3)
    myNumber = read_input()
    student1.add_course(”Basics in Programming”)
    student2.add_course(”Algorithms and Datastructures”)
    myList = (myNumber)
    write_output(myList)
    print(myNumber)
```

Recap:

```
class Student:

    def __init__(self, myName, myNumber):
        self.__name = myName
        self.__id = myNumber
        self.__grades = [ ]
        self.__courses = [ ]

    def add_course(self,course):
        self.__courses.append(course)

main():
    student1 = Student(”Barbara”,123)
    student2 = Student(”Angelina”,564)
    studentRegistry = (student1,student2,student3)
    name = read_input()
    student1.add_course(”Basics in Programming”)
    student2.add_course(”Algorithms and Datastructures”)
```

Module: student
File: student.py

```
class Student:

    def __init__(self, myName, myNumber):
        self.__name = myName
        self.__id = myNumber
        self.__grades = [ ]
        self.__courses = [ ]

    def add_course(self,course):
        self.__courses.append(course)

main():
    student1 = Student("Barbara",123)
    student2 = Student("Angelina",564)
    student3 = Student("Brad", 897)
    studentRegistry = (student1,student2,student3)
    student1.add_course("Basics in Programming")
    student2.add_course("Algorithms and Datastructures")
```


```
import student
main():
    student1 = student.Student(”Barbara”,123)
    student2 = student.Student(”Angelina”,564)
    student1.add_course(”Basics in Programming”)
    student2.add_course(”Algorithms and Datastructures”)
```

Or

```
import student as stud
main():
    student1 = stud.Student(”Barbara”,123)
    student2 = stud.Student(”Angelina”,564)
    student1.add_course(”Basics in Programming”)
    student2.add_course(”Algorithms and Datastructures”)
```

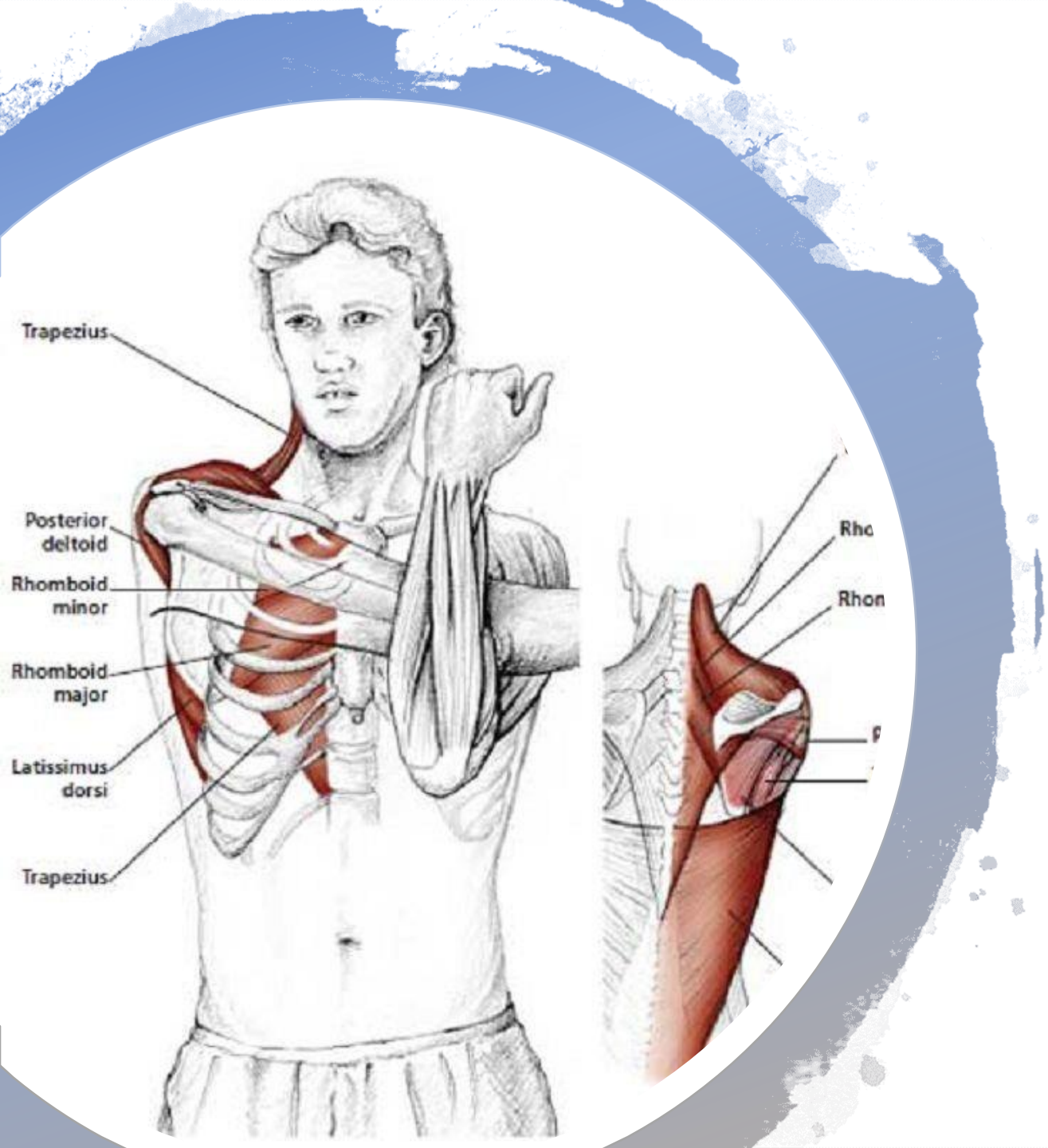

```
import student
main():
    student1 = student.Student(”Barbara”,123)
    student2 = student.Student(”Angelina”,564)
    student1.add_course(”Basics in Programming”)
    student2.add_course(”Algorithms and Datastructures”)
```

Or

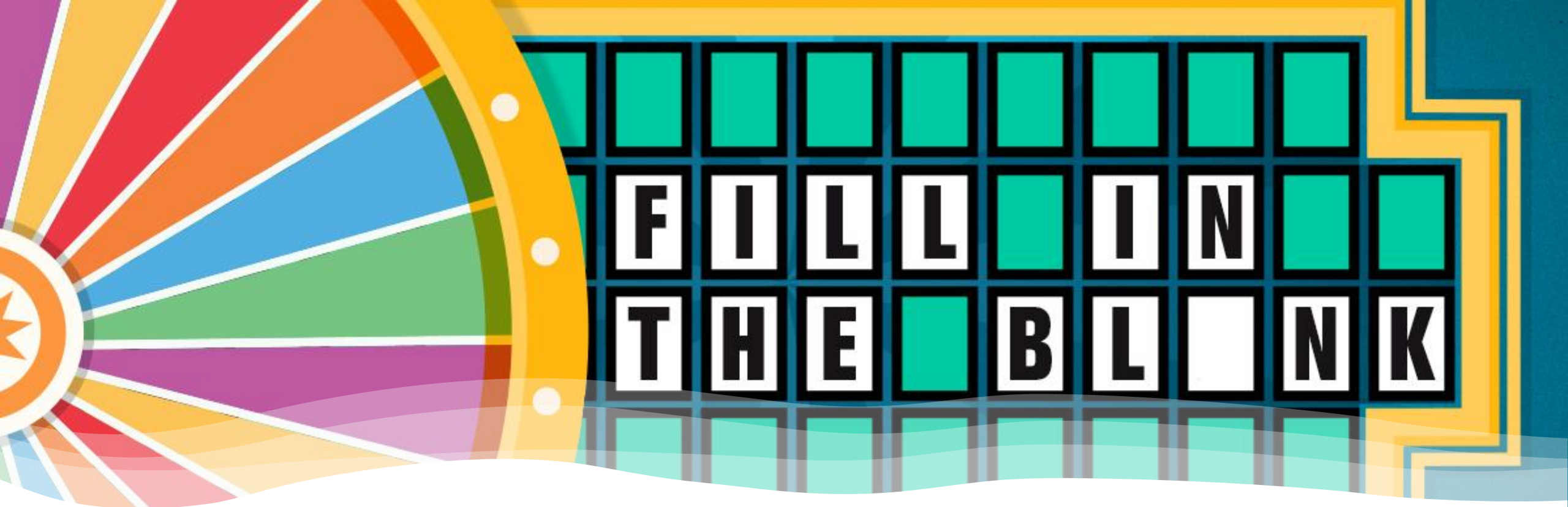
```
import student *
main():
    student1 = Student(”Barbara”,123)
    student2 = Student(”Angelina”,564)
    student1.add_course(”Basics in Programming”)
    student2.add_course(”Algorithms and Datastructures”)
```

Good to Know

- Guidelines:
  - Imports always at the start of the file
  - Import module *
    Be careful: This could potentially hide some things you defined
  - Import is only happening once per interpreter session,
    if you make changes to the module, you need to restart the interpreter
    (or use this: `import importlib; importlib.reload(modulename))`
- There is much more, like packages → not part of this course
- The module does NOT need to be in the same directory, for our purpose it is easier to keep it that way



Break:
Move your Shoulders



Whats up with all the _ _ _ _ _ _ _ _ _ _ _ _ _ _

It is about who has access to what

_ _birthyear hides the attribute “birthyear” from the world outside of your class

student1._ _birthyear = 1999 is **not** valid in your main():

But Why?

It makes it way easier to structure and maintain your code. If there are changes, you only need to update the Class, not every program that uses it.

Example

```
Class Student1:

    def __init__(self, myName):
        self.name = myName
        self.age = 0



    def get_age(self):
        return self.age


main():

    myStudent.age = 15

    ........
    if myStudent.age < 18:

        print("sorry, you are underage")
```

```
Class Student2:

    def __init__(self, myName):
        self.__name = myName
        self.__age = 0

    def get_age(self):
        return self.__age

    def set_age(self, myAge):

        if (-0.75 <myAge<150):
            self.__age = myAge

main():

    myStudent.set_age(15)

     ........
    age = myStudent.get_age()

    if age < 18:

        print("sorry, you are underage")
```

Example

```
Class Student1:
    def __init__(self, myName):
        self.name = myName
        self.age = 0

    def get_age(self):
        return self.age

main():
    myStudent.age = 15
    ........
    if myStudent.age < 18:
        print("sorry, you are underage")
```

Change attribute **age** to **birthyear**

```
Class Student2:
    def __init__(self, myName):
        self.__name = myName
        self.__age = 0

    def get_age(self):
        return self.__age

    def set_age(self, myAge):
        if (-0.75 <myAge<150):
            self.__age = myAge

main():
    myStudent.set_age(15)
    ........
    age = myStudent.get_age()
    if age < 18:
        print("sorry, you are underage")
```

Example

```
Class Student1:

    def __init__(self, myName):
        self.name = myName
        self.birthyear = 0


    def get_age(self):

        age = CUR_YEAR -self.birthyear
        return age


main():

    myStudent.age = 15

    ........
    if myStudent.age < 18:

        print("sorry, you are underage")
```

Change attribute **age** to **birthyear**

```
Class Student2:

    def __init__(self, myName):
        self._ _name = myName
        self._ _birthyear = 0

    def get_age(self):

        age = CUR_YEAR – self._ _birthyear
        return age

    def set_age(self, myAge):

         if (-0.75 <myAge<150):
            self._ _ birthyear = CUR_YEAR - myAge
main():

    myStudent.set_age(15)

     ........
    age = myStudent.get_age()
    if age < 18:

        print("sorry, you are underage")
```

Example

```
Class Student1:

    def __init__(self, myName):
        self.name = myName
        self.birthyear = 0


    def get_age(self):

        age = CUR_YEAR -self.birthyear
        return age


main():

    myStudent.age = 15

    ........
    if myStudent.age < 18:

        print("sorry, you are underage")
```

Change attribute **age** to **birthyear**

```
Class Student2:

    def __init__(self, myName):
        self._ _name = myName
        self._ _birthyear = 0

    def get_age(self):

        age = CUR_YEAR – self._ _birthyear
        return age

    def set_age(self, myAge):

         if (-0.75 <myAge<150):
            self._ _ birthyear = CUR_YEAR - myAge
main():

    myStudent.set_age(15)

     ........
    age = myStudent.get_age()
    if age < 18:

        print("sorry, you are underage")
```

Getters and Setters

Use for **every** attribute set- and get-methods!

```
def set_age(self,myAge):
    self._ _age = myAge

def get_age(self):
    return self._ _ age
```

List Attributes

```
class Student

    def _ _init_ _(self, myName):
        self._ _name = myName
        self._ _grades = [ ]


    def add_grade(self,myGrade):
        if 0 <= myGrade <= 5:
            self._ _grades.add(myGrade)


main():
    student1 = Student("Barbara")

    student1.add_grade(5)
```

?

List Attributes

```
class Student
    def _ _init_ _(self, myName):
        self._ _name = myName
        self._ _grades = [ ]

    def add_grade(self,myGrade):
        if 0 <= myGrade <= 5:
            self._ _grades.add(myGrade)
            return True
        else:
            return False


main():
    student1 = Student("Barbara")
    if student1.add_grade(5):
        print("grade added successfully")
    else:
        print("could not add grade to " + student1.get_name())
```

You like print()?

Do your own for your classes

_ _ methods _ _
cannot be directly called

```
class Student

    def _ _init_ _(self, myName, myNumber):

        self._ _name = myName
        self._ _ number = myNumber
        self._ _grades = [ ]


    def _ _str_ _(self):

        printString = "Student " + self._ _name + ", ID:" + self._ _number
        return printString

main():
    student1 = Student("Barbara",123)
    print(student1)
```

Student Barbara, ID: 123

What is Supposed to be in a Class

_ _ init_ _ ← this is how you get an object of this class

_ _string_ _ ←to make life easier for programmers using your class

(usually) for all attributes:
set_attribute(attribute_value):

get_attribute():

methods that are useful with your object / everyone needs with your object ←Calculate average degree

Good to Know

- If you hide your attributes _ _
  - It is easier to update your class without updating other programs
  - It is cleaner
  - It is easier to ensure, that nothing fishy happens with your attribute (student.age = -5), as one can only set the age with the set_age method and you have control over that
  - It is not really true, that it cannot be accessed from outside your class,
    but it is not as easy
- Use separate set_attribute(value) and get_attribute() for all your attributes
- Use return True/False with lists

Objects in Lists

```
def main():
    studDirectory = ()
    newStudent = Student(”Visa”,568)
    studDirectory.append(newStudent)
    newStudent = Student(”Victoria”,784)
    studDirectory.append(newStudent)
    studDirectory.append(Student(”Taige”,778))
    thisName =studDirectory[1].get_name()

    for person in studDirectory:
        print(person)
```

OO-programming and functional programming can be mixed

```
def main():
    studDirectory = ()
    newStudent = Student(”Visa”,568)
    studDirectory.append(newStudent)
    newStudent = Student(”Victoria”,784)
    studDirectory.append(newStudent)
    studDirectory.append(Student(”Taige”,778))
    thisName =studDirectory[1].get_name()
    bestStudent = findBestInClass(studDirectory, ”Basics in Programming”)
    for person in studDirectory:
        print(person)
```

```
def findBestInClass(studDir,myClass):
    curbestGrad = 0
    for myStudent in studDir:
        if myClass in myStudent.get_courses():
            if myStudent.get_grade() > curbestGrad:
                curBestGrade = myStudent.get_grade()
                curBestStudent = myStudent
    return curBestStudent
```


"That's all Folks!"
lsberg