

Outline

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

MCS 260 Lecture 18
Introduction to Computer Science
Jan Verschelde, 8 October 2007

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

Outline

Modular Design
programming in the large
software engineering

Modules in Python
importing modules
stack of data

Modules in Maple
computing in \mathbb{Z}_5

Summary + Assignments

MCS 260 L-18

8 October 2007

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modular Design

building large software systems

- ▶ we have practiced programming in the small
- ▶ programming in the large requires modular design
- ▶ characteristics of large programs:
 1. size: more than 100,000 lines of code
 2. effort: many teams of programmers
 3. time: program maintenance and evolution
- ▶ modular design of programs aims to control the complexity of a program by dividing it into *modules*
- ▶ a typical example of a module is a library of functions

MCS 260 L-18

8 October 2007

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modular Design

building large software systems

- ▶ we have practiced programming in the small
- ▶ programming in the large requires modular design
- ▶ characteristics of large programs:
 1. size: more than 100,000 lines of code
 2. effort: many teams of programmers
 3. time: program maintenance and evolution
- ▶ modular design of programs aims to control the complexity of a program by dividing it into *modules*
- ▶ a typical example of a module is a library of functions

MCS 260 L-18

8 October 2007

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modular Design

building large software systems

- ▶ we have practiced programming in the small
- ▶ programming in the large requires modular design
- ▶ characteristics of large programs:
 1. size: more than 100,000 lines of code
 2. effort: many teams of programmers
 3. time: program maintenance and evolution
- ▶ modular design of programs aims to control the complexity of a program by dividing it into *modules*
- ▶ a typical example of a module is a library of functions

Modular Design

building large software systems

- ▶ we have practiced programming in the small
- ▶ programming in the large requires modular design
- ▶ characteristics of large programs:
 1. size: more than 100,000 lines of code
 2. effort: many teams of programmers
 3. time: program maintenance and evolution
- ▶ modular design of programs aims to control the complexity of a program by dividing it into *modules*
- ▶ a typical example of a module is a library of functions

Modular Design

building large software systems

- ▶ we have practiced programming in the small
- ▶ programming in the large requires modular design
- ▶ characteristics of large programs:
 1. size: more than 100,000 lines of code
 2. effort: many teams of programmers
 3. time: program maintenance and evolution
- ▶ modular design of programs aims to control the complexity of a program by dividing it into *modules*
- ▶ a typical example of a module is a library of functions

Modular Design

building large software systems

- ▶ we have practiced programming in the small
- ▶ programming in the large requires modular design
- ▶ characteristics of large programs:
 1. size: more than 100,000 lines of code
 2. effort: many teams of programmers
 3. time: program maintenance and evolution
- ▶ modular design of programs aims to control the complexity of a program by dividing it into *modules*
- ▶ a typical example of a module is a library of functions

Software Systems have Layers or Levels

MCS 260 L-18

8 October 2007

Layers typical for almost any software system:

1. the *kernel* consists of basic functions
2. the *main operations* apply the kernel
3. the *user interface* defines how the user interacts with the software

For example, Maple consists of

1. built in functions written in C
2. procedures written in Maple's language
3. command line and worksheet interface

In Maple, to see the code for `gcd`:

```
> interface(verboseproc=3);  
> print(gcd);
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Software Systems have Layers or Levels

MCS 260 L-18

8 October 2007

Layers typical for almost any software system:

1. the *kernel* consists of basic functions
2. the *main operations* apply the kernel
3. the *user interface* defines how the user interacts with the software

For example, Maple consists of

1. built in functions written in C
2. procedures written in Maple's language
3. command line and worksheet interface

In Maple, to see the code for `gcd`:

```
> interface(verboseproc=3);  
> print(gcd);
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Software Systems have Layers or Levels

MCS 260 L-18

8 October 2007

Layers typical for almost any software system:

1. the *kernel* consists of basic functions
2. the *main operations* apply the kernel
3. the *user interface* defines how the user interacts with the software

For example, Maple consists of

1. built in functions written in C
2. procedures written in Maple's language
3. command line and worksheet interface

In Maple, to see the code for `gcd`:

```
> interface(verboseproc=3);  
> print(gcd);
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Software Systems have Layers or Levels

MCS 260 L-18

8 October 2007

Layers typical for almost any software system:

1. the *kernel* consists of basic functions
2. the *main operations* apply the kernel
3. the *user interface* defines how the user interacts with the software

For example, Maple consists of

1. built in functions written in C
2. procedures written in Maple's language
3. command line and worksheet interface

In Maple, to see the code for `gcd`:

```
> interface(verboseproc=3);  
> print(gcd);
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Software Systems have Layers or Levels

MCS 260 L-18

8 October 2007

Layers typical for almost any software system:

1. the *kernel* consists of basic functions
2. the *main operations* apply the kernel
3. the *user interface* defines how the user interacts with the software

For example, Maple consists of

1. built in functions written in C
2. procedures written in Maple's language
3. command line and worksheet interface

In Maple, to see the code for `gcd`:

```
> interface(verboseproc=3);  
> print(gcd);
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Software Systems have Layers or Levels

MCS 260 L-18

8 October 2007

Layers typical for almost any software system:

1. the *kernel* consists of basic functions
2. the *main operations* apply the kernel
3. the *user interface* defines how the user interacts with the software

For example, Maple consists of

1. built in functions written in C
2. procedures written in Maple's language
3. command line and worksheet interface

In Maple, to see the code for `gcd`:

```
> interface(verboseproc=3);  
> print(gcd);
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Software Systems have Layers or Levels

MCS 260 L-18

8 October 2007

Layers typical for almost any software system:

1. the *kernel* consists of basic functions
2. the *main operations* apply the kernel
3. the *user interface* defines how the user interacts with the software

For example, Maple consists of

1. built in functions written in C
2. procedures written in Maple's language
3. command line and worksheet interface

In Maple, to see the code for `gcd`:

```
> interface(verboseproc=3);  
> print(gcd);
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Outline

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

Modules

components of software systems

A software system consists of

1. a collection of modules; and
2. the relations between the modules.

Modular design defines the decomposition of a system into modules. A module can have modular components.

Each module has an **interface** and a **body**:

interface is the set of all elements in a module available to all users of the module, also called the module's **exported resources**

body is what realizes the functionalities of a module, also called the **implementation**.

A module *imports* resources from another module.

A module *exports* resources via its interface.

Modules

components of software systems

A software system consists of

1. a collection of modules; and
2. the relations between the modules.

Modular design defines the decomposition of a system into modules. A module can have modular components.

Each module has an **interface** and a **body**:

interface is the set of all elements in a module available to all users of the module, also called the module's **exported resources**

body is what realizes the functionalities of a module, also called the **implementation**.

A module *imports* resources from another module.

A module *exports* resources via its interface.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modules

components of software systems

A software system consists of

1. a collection of modules; and
2. the relations between the modules.

Modular design defines the decomposition of a system into modules. A module can have modular components.

Each module has an **interface** and a **body**:

interface is the set of all elements in a module available to all users of the module, also called the module's **exported resources**

body is what realizes the functionalities of a module, also called the **implementation**.

A module *imports* resources from another module.

A module *exports* resources via its interface.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modules

components of software systems

A software system consists of

1. a collection of modules; and
2. the relations between the modules.

Modular design defines the decomposition of a system into modules. A module can have modular components.

Each module has an **interface** and a **body**:

interface is the set of all elements in a module available to all users of the module, also called the module's **exported resources**

body is what realizes the functionalities of a module, also called the **implementation**.

A module *imports* resources from another module.

A module *exports* resources via its interface.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modules

components of software systems

A software system consists of

1. a collection of modules; and
2. the relations between the modules.

Modular design defines the decomposition of a system into modules. A module can have modular components.

Each module has an **interface** and a **body**:

interface is the set of all elements in a module available to all users of the module, also called the module's **exported resources**

body is what realizes the functionalities of a module, also called the **implementation**.

A module *imports* resources from another module.

A module *exports* resources via its interface.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modules

components of software systems

A software system consists of

1. a collection of modules; and
2. the relations between the modules.

Modular design defines the decomposition of a system into modules. A module can have modular components.

Each module has an **interface** and a **body**:

interface is the set of all elements in a module available to all users of the module, also called the module's **exported resources**

body is what realizes the functionalities of a module, also called the **implementation**.

A module *imports* resources from another module.

A module *exports* resources via its interface.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modules

components of software systems

A software system consists of

1. a collection of modules; and
2. the relations between the modules.

Modular design defines the decomposition of a system into modules. A module can have modular components.

Each module has an **interface** and a **body**:

interface is the set of all elements in a module available to all users of the module, also called the module's **exported resources**

body is what realizes the functionalities of a module, also called the **implementation**.

A module *imports* resources from another module.

A module *exports* resources via its interface.

Principles of Modular Design

criteria for good software engineering

The software architect designs the system architecture.
The system architecture represents the decomposition of the system into modules and the intermodule relations.

A first recommendation to design modular software:

- ▶ Information Hiding

The interface must be separated from the body.
Programs that rely on the module via its interface do not have to be rewritten as the body changes.

This principle implies that the interface of a module contains the right kind of information.

Users who need to manipulate polynomials should not be required to take into account the internal data structures used to represent the polynomials.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

Principles of Modular Design

criteria for good software engineering

The software architect designs the system architecture.
The system architecture represents the decomposition of the system into modules and the intermodule relations.

A first recommendation to design modular software:

- ▶ Information Hiding

The interface must be separated from the body.
Programs that rely on the module via its interface do not have to be rewritten as the body changes.

This principle implies that the interface of a module contains the right kind of information.

Users who need to manipulate polynomials should not be required to take into account the internal data structures used to represent the polynomials.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

Principles of Modular Design

criteria for good software engineering

The software architect designs the system architecture.
The system architecture represents the decomposition of the system into modules and the intermodule relations.

A first recommendation to design modular software:

- ▶ Information Hiding

The interface must be separated from the body.
Programs that rely on the module via its interface do not have to be rewritten as the body changes.

This principle implies that the interface of a module contains the right kind of information.

Users who need to manipulate polynomials should not be required to take into account the internal data structures used to represent the polynomials.

Bottom-Up Design of Programs

principle of low coupling and high cohesion

Modules are implemented by different teams of programmers, often working over different time periods. The functionality of a module must be ready for testing and verification independently of the rest of the program.

A second recommendation to design modular software:

- ▶ Low Coupling and High Cohesion

Low coupling means that modules are largely independent from each other.

Functions often used together belong to the same module so each module has a high internal cohesion.

A module to manipulate polynomials should collect all the operations needed in the software system.

Bottom-Up Design of Programs

principle of low coupling and high cohesion

Modules are implemented by different teams of programmers, often working over different time periods. The functionality of a module must be ready for testing and verification independently of the rest of the program.

A second recommendation to design modular software:

- ▶ **Low Coupling and High Cohesion**

Low coupling means that modules are largely independent from each other.

Functions often used together belong to the same module so each module has a high internal cohesion.

A module to manipulate polynomials should collect all the operations needed in the software system.

Bottom-Up Design of Programs

principle of low coupling and high cohesion

Modules are implemented by different teams of programmers, often working over different time periods. The functionality of a module must be ready for testing and verification independently of the rest of the program.

A second recommendation to design modular software:

- ▶ **Low Coupling and High Cohesion**

Low coupling means that modules are largely independent from each other.

Functions often used together belong to the same module so each module has a high internal cohesion.

A module to manipulate polynomials should collect all the operations needed in the software system.

Bottom-Up Design of Programs

principle of low coupling and high cohesion

Modules are implemented by different teams of programmers, often working over different time periods. The functionality of a module must be ready for testing and verification independently of the rest of the program.

A second recommendation to design modular software:

- ▶ Low Coupling and High Cohesion

Low coupling means that modules are largely independent from each other.

Functions often used together belong to the same module so each module has a high internal cohesion.

A module to manipulate polynomials should collect all the operations needed in the software system.

Reuse of Modules

standard libraries of components

Software development is an expensive process...

A third recommendation to design modular software:

- ▶ Design for Change

For example, use of parameters and constants for data that may later change.

For modules to manipulate polynomials, we foresee that different coefficient fields could be needed.

Object-oriented design is typically bottom up and leads to reusable software.

We will cover object-oriented programming in Python.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Reuse of Modules

standard libraries of components

Software development is an expensive process...

A third recommendation to design modular software:

- ▶ Design for Change

For example, use of parameters and constants for data that may later change.

For modules to manipulate polynomials, we foresee that different coefficient fields could be needed.

Object-oriented design is typically bottom up and leads to reusable software.

We will cover object-oriented programming in Python.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Reuse of Modules

standard libraries of components

MCS 260 L-18

8 October 2007

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Software development is an expensive process...

A third recommendation to design modular software:

- ▶ Design for Change

For example, use of parameters and constants for data that may later change.

For modules to manipulate polynomials, we foresee that different coefficient fields could be needed.

Object-oriented design is typically bottom up and leads to reusable software.

We will cover object-oriented programming in Python.

Outline

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

MCS 260 L-18

8 October 2007

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

Modules in Python

importing modules

The syntax to import a module is

```
import < module >
```

For example: `import math`.

Then we can compute $\sqrt{2}$ via `math.sqrt(2)`.

If we only need one element of a module:

```
from < module > import < element >
```

For example: `from math import sqrt`.

Then we can compute $\sqrt{2}$ simply as `sqrt(2)`.

If `import math` is successful, then `help(math)` or `help(math.cos)` shows information about the module `math` or the function `math.cos`.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modules in Python

importing modules

The syntax to import a module is

```
import < module >
```

For example: `import math`.

Then we can compute $\sqrt{2}$ via `math.sqrt(2)`.

If we only need one element of a module:

```
from < module > import < element >
```

For example: `from math import sqrt`.

Then we can compute $\sqrt{2}$ simply as `sqrt(2)`.

If `import math` is successful, then `help(math)` or `help(math.cos)` shows information about the module `math` or the function `math.cos`.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modules in Python

importing modules

The syntax to import a module is

```
import < module >
```

For example: `import math`.

Then we can compute $\sqrt{2}$ via `math.sqrt(2)`.

If we only need one element of a module:

```
from < module > import < element >
```

For example: `from math import sqrt`.

Then we can compute $\sqrt{2}$ simply as `sqrt(2)`.

If `import math` is successful, then `help(math)` or `help(math.cos)` shows information about the module `math` or the function `math.cos`.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Outline

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

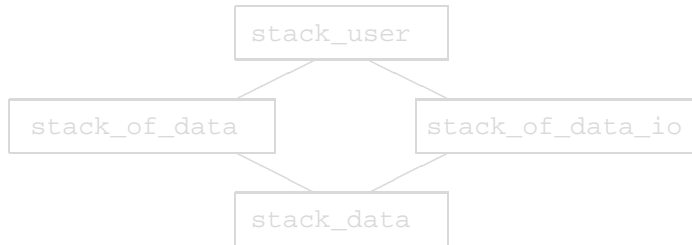
Summary + Assignments

A Stack of Data

bottom up design

Suppose we want a stack of data as data structure.

The bottom up design hides the internal representation from the program `stack_user`. Four `.py` files:



The module `stack_data` is just one line:

```
the_stack = []
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

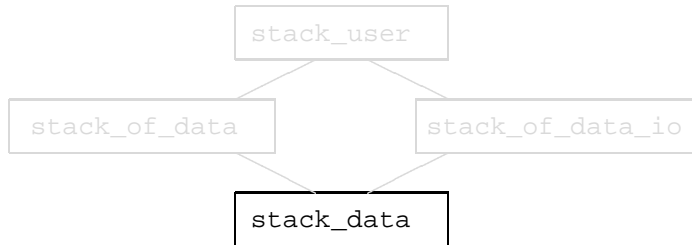
Summary +
Assignments

A Stack of Data

bottom up design

Suppose we want a stack of data as data structure.

The bottom up design hides the internal representation from the program `stack_user`. Four `.py` files:



The module `stack_data` is just one line:

```
the_stack = []
```

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

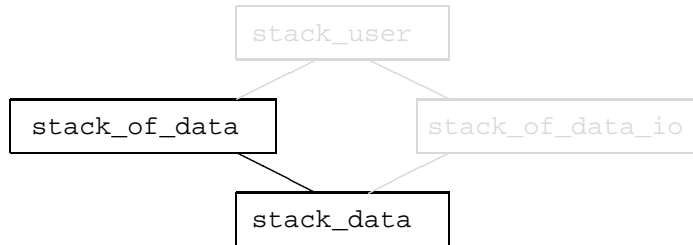
[Summary +
Assignments](#)

A Stack of Data

bottom up design

Suppose we want a stack of data as data structure.

The bottom up design hides the internal representation from the program `stack_user`. Four `.py` files:



The module `stack_data` is just one line:

```
the_stack = []
```

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

[Summary +
Assignments](#)

A Stack of Data

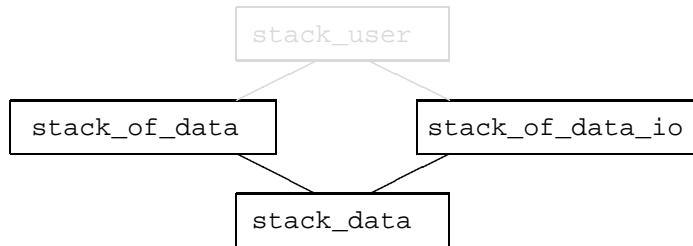
bottom up design

MCS 260 L-18

8 October 2007

Suppose we want a stack of data as data structure.

The bottom up design hides the internal representation from the program `stack_user`. Four `.py` files:



The module `stack_data` is just one line:

```
the_stack = []
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

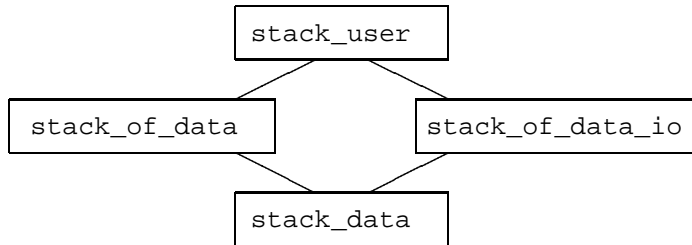
Summary +
Assignments

A Stack of Data

bottom up design

Suppose we want a stack of data as data structure.

The bottom up design hides the internal representation from the program `stack_user`. Four `.py` files:



The module `stack_data` is just one line:

```
the_stack = []
```

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

[Summary +
Assignments](#)

A Stack of Data

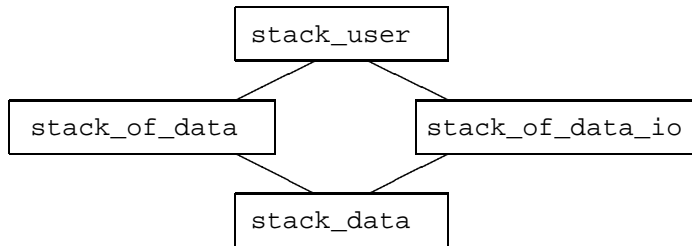
bottom up design

MCS 260 L-18

8 October 2007

Suppose we want a stack of data as data structure.

The bottom up design hides the internal representation from the program `stack_user`. Four `.py` files:



The module `stack_data` is just one line:

```
the_stack = []
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Interface of `stack_of_data`

exported resources

Function definitions in the file `stack_of_data.py`:

```
def push(item):  
    "pushes the item on the stack"  
  
def length():  
    "returns the length of the stack"  
  
def pop():  
    "returns an item off the stack"
```

This is the view offered to `stack_user`.

The user does not know that a list stores the stack.

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

[Summary +
Assignments](#)

Interface of `stack_of_data`

exported resources

Function definitions in the file `stack_of_data.py`:

```
def push(item):  
    "pushes the item on the stack"  
  
def length():  
    "returns the length of the stack"  
  
def pop():  
    "returns an item off the stack"
```

This is the view offered to `stack_user`.

The user does not know that a list stores the stack.

Body of `stack_of_data`

an implementation of the module

```
from stack_data import the_stack

def push(item):
    "pushes the item on the stack"
    the_stack.insert(0,item)

def length():
    "returns the length of the stack"
    return len(the_stack)

def pop():
    "returns an item off the stack"
    return the_stack.pop(0)
```

The first `import` statement is *global*: it makes `the_stack` available to all functions.

For *local* use, move the `import` into a function definition.

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

[Summary +
Assignments](#)

Body of stack_of_data

an implementation of the module

```
from stack_data import the_stack

def push(item):
    "pushes the item on the stack"
    the_stack.insert(0,item)

def length():
    "returns the length of the stack"
    return len(the_stack)

def pop():
    "returns an item off the stack"
    return the_stack.pop(0)
```

The first `import` statement is *global*: it makes `the_stack` available to all functions.

For *local* use, move the `import` into a function definition.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Body of `stack_of_data`

an implementation of the module

```
from stack_data import the_stack

def push(item):
    "pushes the item on the stack"
    the_stack.insert(0,item)

def length():
    "returns the length of the stack"
    return len(the_stack)

def pop():
    "returns an item off the stack"
    return the_stack.pop(0)
```

The first `import` statement is *global*: it makes `the_stack` available to all functions.

For *local* use, move the `import` into a function definition.

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

[Summary +
Assignments](#)

Body of `stack_of_data`

an implementation of the module

```
from stack_data import the_stack

def push(item):
    "pushes the item on the stack"
    the_stack.insert(0,item)

def length():
    "returns the length of the stack"
    return len(the_stack)

def pop():
    "returns an item off the stack"
    return the_stack.pop(0)
```

The first `import` statement is *global*: it makes `the_stack` available to all functions.

For *local* use, move the `import` into a function definition.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Body of stack_of_data

an implementation of the module

```
from stack_data import the_stack

def push(item):
    "pushes the item on the stack"
    the_stack.insert(0,item)

def length():
    "returns the length of the stack"
    return len(the_stack)

def pop():
    "returns an item off the stack"
    return the_stack.pop(0)
```

The first `import` statement is *global*: it makes `the_stack` available to all functions.

For *local* use, move the `import` into a function definition.

Interface of `stack_of_data_io`

exported resources for input and output

The information for `help(stack_of_data_io)`:

```
def input(L):  
    "places elements of L on stack"  
  
def show_all():  
    "shows all elements in the stack"  
  
def show_top():  
    "shows the top element of the stack"
```

This input/output module is necessary, as the user of the the stack does not have access to the list `the_stack`.

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

[Summary +
Assignments](#)

Interface of `stack_of_data_io`

exported resources for input and output

The information for `help(stack_of_data_io):`

```
def input(L):  
    "places elements of L on stack"  
  
def show_all():  
    "shows all elements in the stack"  
  
def show_top():  
    "shows the top element of the stack"
```

This input/output module is necessary, as the user of the the stack does not have access to the list `the_stack`.

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

[Summary +
Assignments](#)

Body of stack_of_data_io

an implementation of the module

```
from stack_data import the_stack

def input(L):
    "places elements of L on stack"
    for item in L:
        the_stack.insert(0,item)

def show_all():
    "shows all elements in the stack"
    print the_stack

def show_top():
    "shows the top element of the stack"
    if len(the_stack) > 0:
        print the_stack[0]
    else:
        print 'the stack is empty'
```

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

[Summary +
Assignments](#)

Body of stack_of_data_io

an implementation of the module

```
from stack_data import the_stack

def input(L):
    "places elements of L on stack"
    for item in L:
        the_stack.insert(0,item)

def show_all():
    "shows all elements in the stack"
    print the_stack

def show_top():
    "shows the top element of the stack"
    if len(the_stack) > 0:
        print the_stack[0]
    else:
        print 'the stack is empty'
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Body of stack_of_data_io

an implementation of the module

```
from stack_data import the_stack

def input(L):
    "places elements of L on stack"
    for item in L:
        the_stack.insert(0,item)

def show_all():
    "shows all elements in the stack"
    print the_stack

def show_top():
    "shows the top element of the stack"
    if len(the_stack) > 0:
        print the_stack[0]
    else:
        print 'the stack is empty'
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Body of stack_of_data_io

an implementation of the module

```
from stack_data import the_stack

def input(L):
    "places elements of L on stack"
    for item in L:
        the_stack.insert(0,item)

def show_all():
    "shows all elements in the stack"
    print the_stack

def show_top():
    "shows the top element of the stack"
    if len(the_stack) > 0:
        print the_stack[0]
    else:
        print 'the stack is empty'
```

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

[Summary +
Assignments](#)

The User Program

MCS 260 L-18

8 October 2007

To test the module `stack_of_data`, we use an interactive program, with structure:

```
import stack_of_data
import stack_of_data_io

def pop_or_push(choice):
    "calls pop and push functions"

def test_input_output():
    "test input/output operations"

while True:
    choice = raw_input('add, remove, or stop ? (a/r/0) ')
    if choice == '0': break
    pop_or_push(choice)
    test_input_output()
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

The User Program

MCS 260 L-18

8 October 2007

To test the module `stack_of_data`, we use an interactive program, with structure:

```
import stack_of_data
import stack_of_data_io

def pop_or_push(choice):
    "calls pop and push functions"

def test_input_output():
    "test input/output operations"

while True:
    choice = raw_input('add, remove, or stop ? (a/r/0) ')
    if choice == '0': break
    pop_or_push(choice)
    test_input_output()
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

The User Program

MCS 260 L-18

8 October 2007

To test the module `stack_of_data`, we use an interactive program, with structure:

```
import stack_of_data
import stack_of_data_io

def pop_or_push(choice):
    "calls pop and push functions"

def test_input_output():
    "test input/output operations"

while True:
    choice = raw_input('add, remove, or stop ? (a/r/0) ')
    if choice == '0': break
    pop_or_push(choice)
    test_input_output()
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

The User Program

MCS 260 L-18

8 October 2007

To test the module `stack_of_data`, we use an interactive program, with structure:

```
import stack_of_data
import stack_of_data_io

def pop_or_push(choice):
    "calls pop and push functions"

def test_input_output():
    "test input/output operations"

while True:
    choice = raw_input('add, remove, or stop ? (a/r/0) ')
    if choice == '0': break
    pop_or_push(choice)
    test_input_output()
```

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modifications to the Stack

definition of `pop_or_push(choice)`

```
def pop_or_push(choice):  
    "calls pop and push functions"  
    if choice == 'a':  
        item = input('Give an item : ')  
        stack_of_data.push(item)  
    elif choice == 'r':  
        item = stack_of_data.pop()  
        print 'item popped : %d' % item  
    else:  
        print 'invalid choice, try again'
```

Input and Output of the Stack

definition of test_input_output()

```
def test_input_output():
    "test input/output operations"
    K = input('give a list : ')
    stack_of_data_io.input(K)
    L = stack_of_data.length()
    print 'length of stack : %d' % L
    print 'printing top element'
    stack_of_data_io.show_top()
    print 'printing the stack'
    stack_of_data_io.show_all()
```

[Modular Design](#)

programming in the large
software engineering

[Modules in Python](#)

importing modules
stack of data

[Modules in Maple](#)

computing in \mathbb{Z}_5

[Summary +
Assignments](#)

Outline

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

MCS 260 L-18

8 October 2007

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary + Assignments

Modules in Maple

computing in \mathbb{Z}_5

$\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$, with plus and times defined via the usual $+$ and \times , followed by modulo 5.

Definition of the module $\mathbb{Z}_5, +, \times$:

```
> z5 := module()
      export plus, times;
      plus := (a,b) -> a + b mod 5;
      times := (a,b) -> a * b mod 5;
end module;
```

Use of the module:

```
> z5:-plus(2,4);
```

1

```
> z5:-times(2,4);
```

3

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modules in Maple

computing in \mathbb{Z}_5

$\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$, with plus and times defined via the usual $+$ and \times , followed by modulo 5.

Definition of the module $\mathbb{Z}_5, +, \times$:

```
> z5 := module()
      export plus, times;
      plus := (a,b) -> a + b mod 5;
      times := (a,b) -> a * b mod 5;
end module;
```

Use of the module:

```
> z5:-plus(2,4);
```

1

```
> z5:-times(2,4);
```

3

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Modules in Maple

computing in \mathbb{Z}_5

$\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$, with plus and times defined via the usual $+$ and \times , followed by modulo 5.

Definition of the module $\mathbb{Z}_5, +, \times$:

```
> z5 := module()
      export plus, times;
      plus := (a,b) -> a + b mod 5;
      times := (a,b) -> a * b mod 5;
end module;
```

Use of the module:

```
> z5:-plus(2,4);
```

1

```
> z5:-times(2,4);
```

3

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments

Summary + Assignments

MCS 260 L-18

8 October 2007

We covered in the lecture:

- ▶ start of chapter 6 in *Making Use of Python*;
- ▶ chapter 22 in *The Art & Craft of Computing*.

Assignments:

1. Modify the modular design of the stack into a module to represent a queue of data. Provide the operations `enqueue` and `dequeue` in the module `queue_of_data` to respectively add and remove elements. Define input and output and test the module interactively in a program.
2. Define a stack module in Maple.

Homework will be collected on Monday 15 October.

Modular Design

programming in the large
software engineering

Modules in Python

importing modules
stack of data

Modules in Maple

computing in \mathbb{Z}_5

Summary +
Assignments