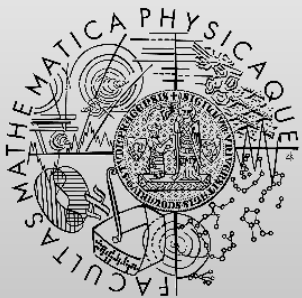


NPRG065: Programming in Python

Lecture 4

<http://d3s.mff.cuni.cz>



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Tomas Bures

Jan Kofron

`{bures,kofron}@d3s.mff.cuni.cz`

Modules

- Module ~ a file with Python definitions
 - extension .py
- To use elements from a module in another module – **import**

All the files with examples till now have been modules

`fibonacci.py`

```
def fib(n):  
    print "computing and printing Fibonacci numbers"
```

`program.py`

```
import fibonacci  
  
fibonacci.fib(5)
```

Import

- Objects in different modules **must be always imported**
 - unlike in Java, cannot be used directly
- Can be placed anywhere in code
- Multiple options of usage

```
import sys
print(sys.argv[0])    # usage with module name

from sys import argv
print(argv[0])        # imported to local namespace

from sys import argv as sysargv
print(sysargv[0])     # imported to local namespace and
                     # renamed

from sys import *     # everything from the sys module
print(argv[0])        # imported to local namespace
```

Modules

- **from sys import ***
 - imports all names except those beginning with an underscore
 - reminder – a name beginning with underscore ~ a special name
- **Module search path**
 - three locations
 - current directory +
 - PYTHONPATH + 
 - installation defaults
 - available through in **sys.path**

Environment variable

Modules

- Module name

```
import fibo
print(fibo.__name__)
```

- When module executed as script

- i.e., python my_module.py
- the name set to "__main__"

```
def fib(n):
    print "computing and printing Fibonacci numbers"

if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

fibo.py

Importable and
executable module

dir()

- Built-in function
- Returns names in a module

```
import fibo
print(dir(fibo))  # -> [ fib ]
```

Actually, the list will be longer but other names there are special

- Without argument – names in local namespace

```
a = [1, 2, 3, 4, 5]
import fibo
fib = fibo.fib
dir() # ['__annotations__', '__builtins__',
        '__doc__', '__loader__', '__name__',
        '__package__', '__spec__', 'a', 'fib',
        'fibo']
```

Packages

package

package

module

```
import sound.effects.echo
```

- Package ~ directory with the `__init__.py` file
 - `__init__.py` is mandatory

sound/	Top-level package
__init__.py	Initialize the sound package
formats/	Subpackage for file format
__init__.py	
wavread.py	
wavwrite.py	
...	
effects/	Subpackage for sound effects
__init__.py	
echo.py	
...	
filters/	Subpackage for filters
__init__.py	
equalizer.py	
...	

Packages

- Importing

```
import sound.effects.echo
sound.effects.echo.echofilter(4)      # full name required

from sound.effects import echo
echo.echofilter(4)                    # module name only

from sound.effects.echo import echofilter
echofilter(4)                         # function name only
```

- Importing * from a package

- only those declared in the variable `__all__` in `__init__.py`

```
sound/effects/__init__.py
```

```
__all__ = ["echo", "surround", "reverse"]
```


Packages

- Naming
 - no conventions as in Java (i.e., like reversed internet name)
 - “pick memorable, meaningful names that aren’t already used on PyPI”
- Conflicting names – no big deal
 - we have renaming
from abc import xyz as mno
 - we can import anywhere in the code

Basic I/O and Exceptions

print

- **`print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`**
 - all *objects* printed separated by *sep* and followed by *end*
 - file – an object where to print
- **`sys.stdout, sys.stderr`**
- **`input([prompt])`**
 - reads a line from input (stripping a trailing newline)
- **`getpass.getpass(prompt='Password: ')`**
 - like `input()` but without echoing
- **`repr(object)`**
 - a printable representation of an object
 - like in the interactive Python shell

Reading and Writing Files

- `open(file, mode='r')`

- returns a file object
 - the actual type may differ based on mode, etc.
 - and thus not all the methods below may be always available

Here are more named parameters
but commonly unused

- file object methods/fields

- `read(size=-1)`
- `write(str_or_bytes)`
- `close()`
- `readline()`
- `readlines()`
- `seek(offset)`
- `readable()`, `writable()`, `seekable()`
- `closed`
- ...

Path-like object
(str, bytes, an object
implementing the `os.PathLike`
protocol)

Reading and Writing Files

- `open(file, mode='r')`

- returns a file object
 - the actual type may differ based on mode, etc.
 - and thus not all the methods below may be available

- file object methods/fields

- `read(size=-1)`
- `write(str_or_bytes)`
- `close()`
- `readline()`
- `readlines()`
- `seek(offset)`
- `readable()`, `writable()`, `seekable()`
- `closed`
- ...

Here are more named parameters but commonly unused

Path-like object
(str, bytes, an object implementing the `os.PathLike` protocol)

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)

Exception

- Represents errors
 - any

```
while True print('Hello world')
    # results in the SyntaxError exception

10 * (1/0)
    # results in the ZeroDivisionError exception

4 + spam*3
    # results in the NameError exception

'2' + 2
    # results in the TypeError exception
```

Try these in the interactive
shell

Exceptions

- Exceptions are either caught or terminates program execution
 - if the exception is not caught in a block where it occurs, it propagates to the upper block
 - if the exception is not caught in a function, it propagates to the calling function
 - if the exception reaches “main” and it not caught, it terminates the program
 - information about the exception is printed
- No need to explicitly declare or catch exceptions
 - like in Java

See `not_caught.py`

Handling exceptions

- try/except/else/finally command

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print('Not a number. Try again...')
```

- except can caught multiple exceptions

```
except (RuntimeError, TypeError, NameError):
    print('An exception occurred:')
```


Handling exceptions

- **else** clause

- executed if no exception occurs
- must follow all **except** clauses

```
import sys

try:
    f = open(sys.argv[1], 'r')
except OSError:
    print('cannot open', sys.argv[1])
else:
    print('File has', len(f.readlines()), 'lines')
    f.close()
```

Handling exceptions

- using the exception object

```
import sys

try:
    f = open(sys.argv[1], 'r')
except OSError as ex:
    print('cannot open', sys.argv[1])
    print(ex)
else:
    print('File has', len(f.readlines()), 'lines')
    f.close()
```



See `exception_info.py`

Exception handling

- multiple **except** clauses

```
try:  
    # code here  
except RuntimeError:  
    print('RuntimeError exception occurred')  
except TypeError:  
    print('TypeError exception occurred')
```

- **finally** clause
 - always executed

```
def divide(x, y):  
    try:  
        result = x / y  
    except ZeroDivisionError:  
        print("division by zero!")  
    else:  
        print("result is", result)  
    finally:  
        print("executing finally clause")
```

Try division.py with
different arguments



The slides are licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).