

NPRG065: Programming in Python *Lecture 9*

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

Department of
Distributed and
Dependable
Systems



Tomas Bures

Petr Hnetynka

{bures, hnetynka}@d3s.mff.cuni.cz



CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Class Design Example – SIS

- “Student Information System”
 - Domain relationships
 - A course has description which is valid for several years
 - Name, e-credits
 - The way it is taught
 - e.g. 2 hours lecture + 2 hours labs once a week, 4 hours labs once per two weeks, 2 x 2 hours lecture + 2 hours labs once a week
 - Each year the course is scheduled to different slots and for different number of parallel groups
 - Each course slot may be taught by a different teacher
 - A student registers to a course and which lecture and labs slots in the schedule he/she will attend
 - Use-cases
 - List courses taught in a particular year
 - Allow student to register to a course
 - List students that are registered to the course but which have not selected slots they will attend
 - Show student schedule
 - Show statistics per teacher (which courses, number of teaching hours, number of students)

57:18 Sestavy Výkazy Chybějící údaje Podezřelé předměty Statistika Mezifakultní studium Přístupy Změna ak.roku

Výuka Vytíženost Agregovaná vytíženost Všechny katedry Diplomanti Archiv vytíženosti Výkony

Katedra a semestr

Katedra: 32-KDSS Semestr: letní Zobraz

CSV

Ústav	Učitel	Kód	Název	Typ	Podíl hod	Celkem hodin	Podíl stud	Celkem stud	Z toho mimofak	Bc	D
32-KDSS	Al Ali Rima	NPRG021	Pokročilé programování na platformě Java	X	28	112	9	52	0		
32-KDSS	Aschenbrenner Vojtěch	NPRG065	Programování v Pythonu	X	28	84	14	50	0		
32-KDSS	Aschenbrenner Vojtěch	NSWI095	Úvod do UNIXu	X	56	280	28	201	0		
32-KDSS	Bulej Lubomír									1	
32-KDSS	Bulej Lubomír	NSWI143	Architektura počítačů	P	56	56	92	92	0		
32-KDSS	Bulej Lubomír	NPRG043	Doporučené postupy v programování	P	28	28	26	26	0		
32-KDSS	Bureš Tomáš									1	2
32-KDSS	Bureš Tomáš	NPRG065	Programování v Pythonu	P	14 / 28	28	24,5 / 49	50	0		
32-KDSS	Bureš Tomáš	NSWI054	Softwarové inženýrství pro spolehlivé systémy	X	29,6	30	8	8	0		
32-KDSS	Bureš Tomáš	NSWE001	Vestavěné systémy a systémy reálného času	P	28	28	6	6	0		
32-KDSS	Bureš Tomáš	NSWE001	Vestavěné systémy a systémy reálného času	X	28	56	6	6	0		
32-KDSS	Čepelík David	NPRG065	Programování v Pythonu	X	28	84	25	50	0		
32-KDSS	Dort Vlastimil	NSWI080	Middleware	X	14	28	13	13	0		
32-KDSS	Hnětynka Petr									1	1
32-KDSS	Hnětynka Petr	NPRG021	Pokročilé programování na platformě Java	P	56	56	52	52	0		
32-KDSS	Hnětynka Petr	NPRG021	Pokročilé programování na platformě Java	X	28	112	20	52	0		
32-KDSS	Hnětynka Petr	NPRG065	Programování v Pythonu	P	14 / 28	28	24,5 / 49	50	0		
32-KDSS	Hnětynka Petr	NSWI058	Výběrový seminář z distribuovaných a komponentových systémů II	X	37,2 / 56	56	2 / 3	3	0		
32-KDSS	Horký Vojtěch	NPRG043	Doporučené postupy v programování	X	56	56	26	26	0		
32-KDSS	Horký Vojtěch	NSWI131	Vyhodnocování výkonnosti počítačových systémů	X	7 / 14	14	4 / 8	8	0		
32-KDSS	Hornáček Adam	NPRG021	Pokročilé programování na platformě Java	X	28	112	23	52	0		
32-KDSS	Houška Petr	NPRG038	Pokročilé programování pro .NET I	X	28	168		107	1		
32-KDSS	Ježek Pavel									6	1
32-KDSS	Ježek Pavel	NPRG038	Pokročilé programování pro .NET I	P	28	28	107	107	1		
32-KDSS	Ježek Pavel	NPRG038	Pokročilé programování pro .NET I	X	28	168	52	107	1		
32-KDSS	Ježek Pavel	NPRG057	Pokročilé programování pro .NET II	P	28	28	36	36	0		
32-KDSS	Ježek Pavel	NPRG064	Programování uživatelských rozhraní v .NET	X	28	28	50	50	1		

Python protocols

- Protocol ~ structural interface
 - a collection of methods an object has to support to implement *something*
- Example – iteration protocol
 - for works with anything iterable

```
for i in anything_iterable:  
    print(i)
```

- iterable ~ has the `__iter__()` method, which returns an object supporting the iteration protocol, i.e., an object with methods
 - `__iter__()` – returns itself
 - `__next__()` – returns the next item or raises the `StopIteration` exception

Protocols

- Many protocols
 - e.g. in `collections.abc` module

ABC	Inherits from	Abstract Methods	Mixin Methods
<code>Container</code>		<code>__contains__</code>	
<code>Hashable</code>		<code>__hash__</code>	
<code>Iterable</code>		<code>__iter__</code>	
<code>Iterator</code>	<code>Iterable</code>	<code>__next__</code>	<code>__iter__</code>
<code>Reversible</code>	<code>Iterable</code>	<code>__reversed__</code>	
<code>Generator</code>	<code>Iterator</code>	<code>send</code> , <code>throw</code>	<code>close</code> , <code>__iter__</code> , <code>__next__</code>
<code>Sized</code>		<code>__len__</code>	
<code>Callable</code>		<code>__call__</code>	
<code>Collection</code>	<code>Sized</code> , <code>Iterable</code> , <code>Container</code>	<code>__contains__</code> , <code>__iter__</code> , <code>__len__</code>	
<code>Sequence</code>	<code>Reversible</code> , <code>Collection</code>	<code>__getitem__</code> , <code>__len__</code>	<code>__contains__</code> , <code>__iter__</code> , <code>__reversed__</code> , <code>index</code> , and <code>count</code>
<code>MutableSequence</code>	<code>Sequence</code>	<code>__getitem__</code> , <code>__setitem__</code> , <code>__delitem__</code> , <code>__len__</code> , <code>insert</code>	Inherited <code>Sequence</code> methods and <code>append</code> , <code>reverse</code> , <code>extend</code> , <code>pop</code> , <code>remove</code> , and <code>__iadd__</code>
<code>ByteString</code>	<code>Sequence</code>	<code>__getitem__</code> , <code>__len__</code>	Inherited <code>Sequence</code> methods
<code>Set</code>	<code>Collection</code>	<code>__contains__</code> , <code>__iter__</code> , <code>__len__</code>	<code>__le__</code> , <code>__lt__</code> , <code>__eq__</code> , <code>__ne__</code> , <code>__gt__</code> , <code>__ge__</code> , <code>__and__</code> , <code>__or__</code> , <code>__sub__</code> , <code>__xor__</code> , and <code>isdisjoint</code>
<code>MutableSet</code>	<code>Set</code>	<code>__contains__</code> , <code>__iter__</code> , <code>__len__</code> , <code>add</code> , <code>discard</code>	Inherited <code>Set</code> methods and <code>clear</code> , <code>pop</code> , <code>remove</code> , <code>__ior__</code> , <code>__iand__</code> , <code>__ixor__</code> , and <code>__isub__</code>

- `__amethod__()` methods called “*special*”
 - <https://docs.python.org/3/reference/datamodel.html#special-method-names>

Special methods

- **`__del__(self)`**
 - finalizer
 - called when the instance is about to be destroyed (by GC)
 - not guaranteed to be called
 - when the interpreter terminates
 - raised exception are ignored
 - only logged to `sys.stderr`
- **Module `gc`**
 - interacting with GC
 - static methods only
 - `gc.collect()` – runs collections
 - `gc.enable()`, `gc.disable()`, `gc.isenabled()`, ...

See
[finalizer.py](#)

Special methods

See
[tostring.py](#)

- **`__repr__(self)`**
 - returns the “official” string representation of an object
 - should look like a valid Python expression that could be used to recreate an object with the same value
 - called by the **`repr()`** built-in function
- **`__str__(self)`**
 - returns the “informal” or nicely printable string representation of an object
 - called by the built-in functions **`str()`**, **`format()`** and **`print()`**
 - default implementation calls **`__repr__()`**

Special methods: Operators

- Predefined set of operators with defined behavior and syntax.
- Possibility to override behavior by providing custom implementation of the matching special method in the class. Except **and**, **is**, and **or**.
- It is not possible to change the syntax or add new operators.
- For arithmetic operators there are three types of special methods:
 - “Normal” i.e.
 - `object.__add__(self, other)`
 - Self is left operand, other is right operand.
 - “Reverse” i.e.
 - `object.__radd__(self, other)`
 - Self is right operand, other is left operand. If defined takes precedence to “normal”
 - “In-place” i.e.
 - `object.__iadd__(self, other)`
 - Used by `+=` syntax. If possible modify self object in-place

Operators - arithmetic

Operator	Method	Reverse argument method
+	object.__add__(self, other)	object.__radd__(self, other)
-	object.__sub__(self, other)	object.__rsub__(self, other)
*	object.__mul__(self, other)	object.__rmul__(self, other)
@	object.__matmul__(self, other)	object.__rmatmul__(self, other)
/	object.__truediv__(self, other)	object.__rtruediv__(self, other)
//	object.__floordiv__(self, other)	object.__rfloordiv__(self, other)
%	object.__mod__(self, other)	object.__rmod__(self, other)
**	object.__pow__(self, other[, mod])	object.__rpow__(self, other)
<<	object.__lshift__(self, other)	object.__rlshift__(self, other)
>>	object.__rshift__(self, other)	object.__rrshift__(self, other)
&	object.__and__(self, other)	object.__rand__(self, other)
^	object.__xor__(self, other)	object.__rxor__(self, other)
	object.__or__(self, other)	object.__ror__(self, other)
- (unary)	object.__neg__(self)	
+ (unary)	object.__pos__(self)	
~	object.__invert__(self)	



Operators – in-place arithmetic



Operator	Method
<code>+=</code>	<code>object.__iadd__(self, other)</code>
<code>-=</code>	<code>object.__isub__(self, other)</code>
<code>*=</code>	<code>object.__imul__(self, other)</code>
<code>@=</code>	<code>object.__imatmul__(self, other)</code>
<code>/=</code>	<code>object.__itruediv__(self, other)</code>
<code>//=</code>	<code>object.__ifloordiv__(self, other)</code>
<code>%=</code>	<code>object.__imod__(self, other)</code>
<code>**=</code>	<code>object.__ipow__(self, other[, mod])</code>
<code><<=</code>	<code>object.__ilshift__(self, other)</code>
<code>>>=</code>	<code>object.__irshift__(self, other)</code>
<code>&=</code>	<code>object.__iand__(self, other)</code>
<code>^=</code>	<code>object.__ixor__(self, other)</code>
<code> =</code>	<code>object.__ior__(self, other)</code>



Operators – comparison



Operator	Method
<	object.__lt__(self, other)
<=	object.__le__(self, other)
==	object.__eq__(self, other)
!=	object.__ne__(self, other)
>=	object.__ge__(self, other)
>	object.__gt__(self, other)

Notes:

- negated `__eq__` is used when `__ne__` is not implemented
- `__lt__` on the second argument is used if the first does not implement `__gt__` and vice versa similar for `__le__` and `__ge__`

See
operators-*.py

Special methods

- **`__hash__(self)`**

- returns a hashcode of the object
- int
- called by the builtin function `hash()`
- used in dict, set,...
- recommended implementation – hash from tuple of fields

```
def __hash__(self):  
    return hash((self.name, self.nick, self.color))
```

- implement `__hash__()` only on immutable objects that have also `__eq__()` and will be used as keys in dict and similar

- **`__bool__(self)`**

- conversion to bool value
- e.g., for usage in conditions

See
[hashcode.py](#)

Special methods

- `__call__(self, [arg1, ...])`
 - called when the instance is “called” as a function
 - if this method is defined, `x(arg1, arg2, ...)` is a shorthand for `x.__call__(arg1, arg2, ...)`
- `with` statement

```
with open('workfile') as f:  
    // do something with read data  
print(f.closed) // f is closed automatically
```

- a context manager – an object that defines the runtime context to be established when executing a `with` statement
- `object.__enter__(self)`
 - called at with start
 - `with` binds the method’s return value to the target specified in the `as` clause
- `object.__exit__(self, exc_type, exc_value, traceback)`
 - called when `with` terminates

See
[context-manager.py](#)

Special methods: collections.abc

- Iterable
 - `__iter__(self)` – we already know
- Reversible
 - `__reversed__(self)`
 - returns iterator iterating in reversed order
 - called by `reversed()` builtin
- Sized
 - `__len__(self)`
 - returns lens of the object (e.g., number of item in the container)
 - called by `len()`
 - plus, an object that doesn't define a `__bool__()` method and whose `__len__()` method returns zero is considered to be false in a Boolean context
- Container
 - `__contains__(self, item)`
 - returns true if item is in self

Special methods: collections.abc

- `__getitem__(self, key)`
 - called to implement evaluation of `self[key]`
- `__setitem__(self, key, value)`
 - assignment to `self[key]`
- `__delitem__(self, key)`
 - deletion of `self[key]`
- `__missing__(self, key)`
 - called by `__getitem__()` to implement `self[key]` if key is missing

See
container.py



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