NPRG065: Programming in Python
Lecture 10

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems D3S

FACULTY OF MATHEMATICS AND PHYSICS
Charles University

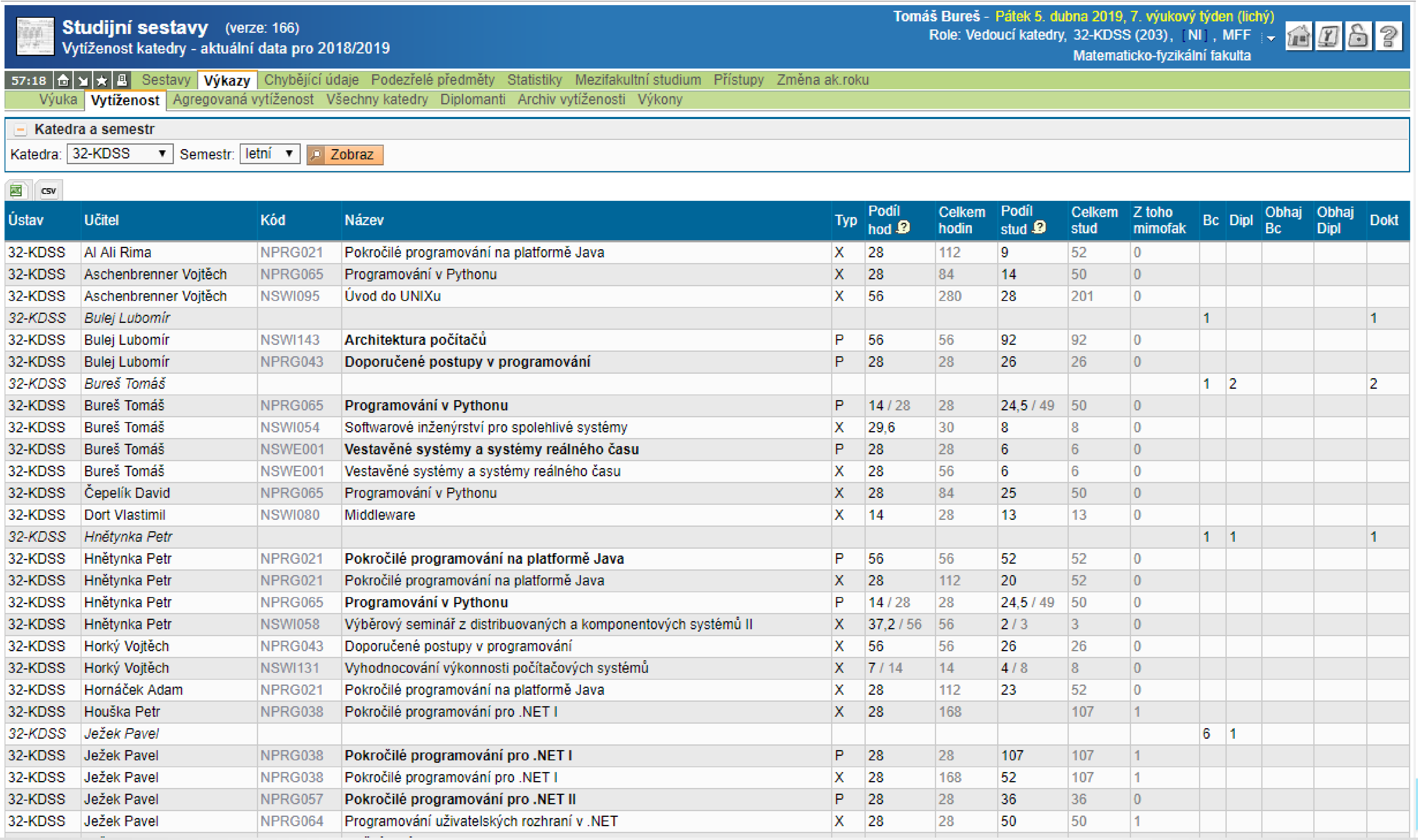
Tomas Bures

Jan Kofron

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

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 students 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)



Ústav	Učitel	Kód	Název	Typ	Podíl hod	Celkem hodin	Podíl stud	Celkem stud	Z toho mimofak	Bc	Dipl	Obhaj Bc	Obhaj Dipl	Dokt
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				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			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			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

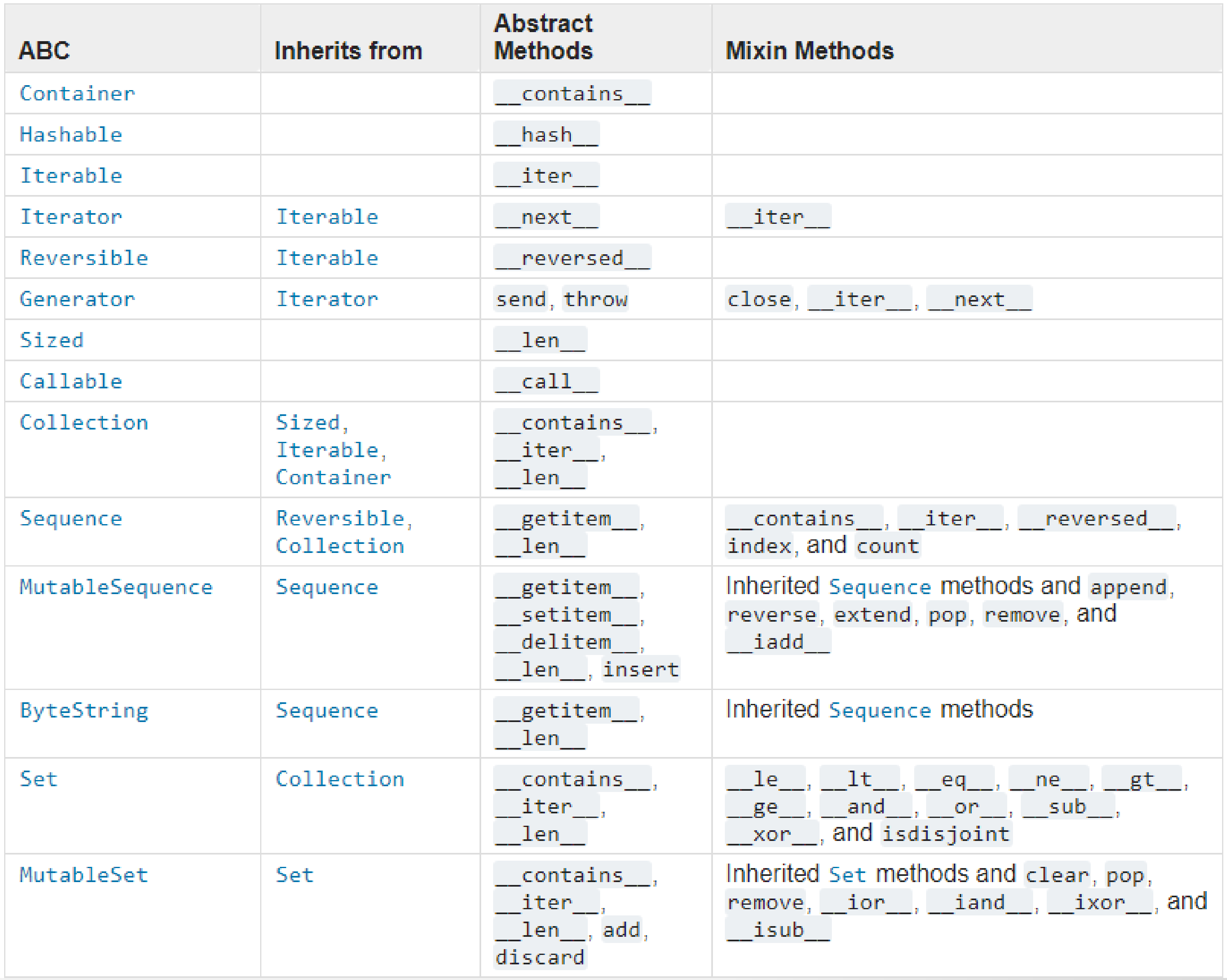
```
for i in anything_iterable:
    print(i)
```

  - 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

Department of Distributed and Dependable Systems

Protocols

- Many protocols
  - e.g. in collections.abc module
- __amethod__() methods called *“special”*
  - https://docs.python.org/3/reference/datamodel.html#special-method-names



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

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

- **`__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__()`**

See
tostring.py

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
+=	object.__**iadd**__(self, other)
-=	object.__**isub**__(self, other)
***=**	object.__**imul**__(self, other)
@=	object.__**imatmul**__(self, other)
/=	object.__**itruediv**__(self, other)
//=	object.__**ifloordiv**__(self, other)
%=	object.__**imod**__(self, other)
****=**	object.__**ipow**__(self, other[, mod])
<<=	object.__**ilshift**__(self, other)
>>=	object.__**irshift**__(self, other)
&=	object.__**iand**__(self, other)
^=	object.__**ixor**__(self, other)
|=	object.__**ior**__(self, other)

Department of Distributed and Dependable Systems D3S

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

Department of Distributed and Dependable Systems

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

Department of Distributed and Dependable Systems

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 continer)
    - 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

Special methods: collections.abc

- Container
  - `__contains__(self, item)`
    - returns true if item is in self
  - `__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]

See
container.py