PyConUS 2025

Pittsburgh

What is the magic of magic methods in the Python language?

Paweł Żal, OpenEDG

PyConUS 2025

Tutorial resources

To get all resources (exercises, presentation, Jupyter notebooks) visit:

`https://tinyurl.com/PyMagic2025`

PyConUS 2025

Install requirements

If you’ve not installed them so far (but it’s optional)

In order to execute all source code from this tutorial, remember to install requirements first by issuing the command:

```
# > pip install -r requirements.txt
```

What is a Magic Method?

A special method, also known as a **magic** method or **dunder** method, is a method whose name begins and ends with a **d**ouble **under**score (hence **dunder**), eg. `__add__()`

Python automatically invokes magic methods in response to certain operations, such as class instantiation, getting object representation, operator overloading, sequence indexing, attribute managing, and many more.

Operator Overloading, Object Representation

1. Start Jupyter Notebook:

```
> jupyter notebook
```

2. Open the `notebooks/notebook01.ipynb`

Key takeaways: Object representation

- the `__str__()` provides the informal string representation of an object, aimed at the user. It is invoked by `print()`, `format()` or `str()`.

Interesting fact: for those functions, if `__str__()` implementation is not provided, then the `__repr__()` will be invoked.

- the `__repr__()` provides the string representation of an object, aimed at the developer providing technical details of the object. It is invoked by `repr()` function.

If `__repr__()` is not provided, general information is returned (class name, memory address).

Key takeaways: Operator overloading

Magic methods support Python following operators:

- arithmetic,
- comparison,
- membership,
- bitwise, and augmented operators.

Arithmetic operator overloading

Operator	Magic Method
+	`__add__(self, other)`
-	`__sub__(self, other)`
*	`__mul__(self, other)`
/	`__truediv__(self, other)`
//	`__floordiv__(self, other)`
%	`__mod__(self, other)`
**	`__pow__(self, other[, modulo])`

Augmented Assignment operator overloading

Short recap:
Augmented assignment operator allows replacement of a statement where operator takes a variable as one of its arguments and assigns the result back to the same variable.

Example:

```
some_value = some_value * 1.2 # regular assignment
some_value *= 1.2 # augmented assignment
```

Augmented Assignment operator overloading

Operator	Magic Method (note „i” in method name)
`+=`	`__iadd__(self, other)`
`-=`	`__isub__(self, other)`
`*=`	`__imul__(self, other)`
`/=`	`__itruediv__(self, other)`
`//=`	`__ifloordiv__(self, other)`
`%=`	`__imod__(self, other)`
`**=`	`__ipow__(self, other[, modulo])`

Right-Hand Arithmetic operator overloading

Consider the following expression:

```
object1 + object2
```

`object1` is a left-hand operand
`object2` is a right-hand operand
`+` is an operator

Right-Hand Arithmetic operator overloading

Operator	Right-Hand Magic Method (thus „r” in method name)
+	__radd__(self, other)
-	__rsub__(self, other)
*	__rmul__(self, other)
/	__rtruediv__(self, other)
//	__rfloordiv__(self, other)
%	__rmod__(self, other)
**	__rpow__(self, other[, modulo])

Unary operator overloading

Operator	Magic Method
+	`__pos__(self)`
-	`__neg__(self)`
abs()	`__abs__(self)`
~	`__invert__(self)`

Comparison operator overloading

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

:)

You may need some break and rest, otherwise we may run into problems during the PyCon 2025 conference. We're just taking care of you and us.

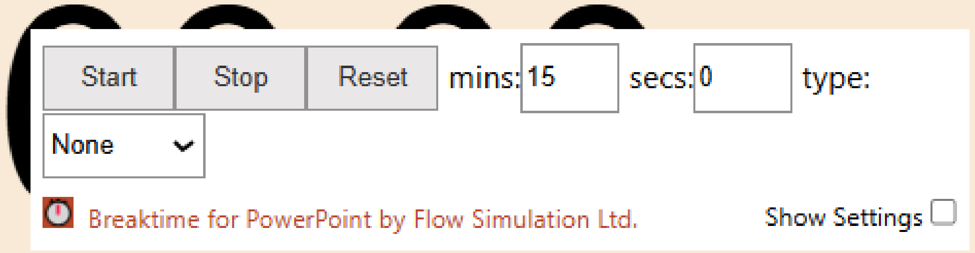
1% complete

For more information about this BSOD generator visit https://bsodmaker.net/

Take a rest for next 15 minutes

A 15 minute break

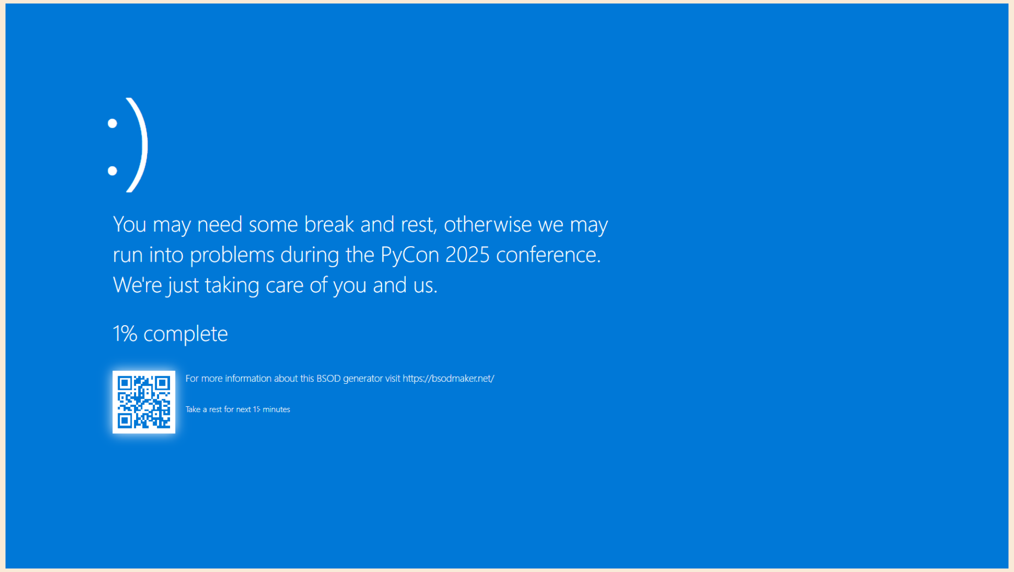
We'll be back in



mins: 15 secs: 0 type: None

Breaktime for PowerPoint by Flow Simulation Ltd.

Show Settings ☐



:)

You may need some break and rest, otherwise we may run into problems during the PyCon 2025 conference. We're just taking care of you and us.

1% complete

For more information about this BSOD generator visit https://bsodmaker.net/

Take a rest for next 15 minutes

Try using magic methods

1. Open the `notebooks/tasks.ipynb`
2. `Focus on Task #1`

Object introspection

Magic methods handle introspection in your custom classes.

It means controlling the objects’ behavior when objects are inspected using built-in functions; could be used for limiting, logging, data enriching etc.

Object introspection

Open the `notebooks/notebook02.ipynb`

Object introspection

Method	Responsibility
`__dir__(self)`	Returns a list of attributes and methods of an object
`__instancecheck__(self, instance)`	Checks whether an object is an **instance** of a certain class
`__subclasscheck__(self, subclass)`	Checks whether a class is a **subclass** of a certain class
`__hasattr__(self, name)`	Checks whether an object has a specific **attribute**

Object lifecycle and customization

Method	Responsibility
`__new__(cls[,...])`	Called to create a new instance of class cls
`__init__(self[,...])`	Called after the instance has been created (by `__new__()`), but before it is returned to the caller
`__del__(self)`	Called when the instance is about to be destroyed

Controlling attribute access

Method	Responsibility
`__getattribute__(self, name)`	Runs when you access an attribute called `name`
`__getattr__(self, name)`	Runs when you access an attribute that doesn't exist in the current object
`__setattr__(self, name, value)`	Runs when you assign value to the attribute called `name`
`__delattr__(self, name)`	Runs when you delete the attribute called `name`

Making the object callable

Method	Responsibility
`__call__(self, *args, **kwargs)`	Called when the instance is „called” as a function

Support for Context Managers

If you want to create a context manager or add context manager functionality to an existing class, then you need to deliver two magic methods:

- `__enter__()`
- `__exit__()`

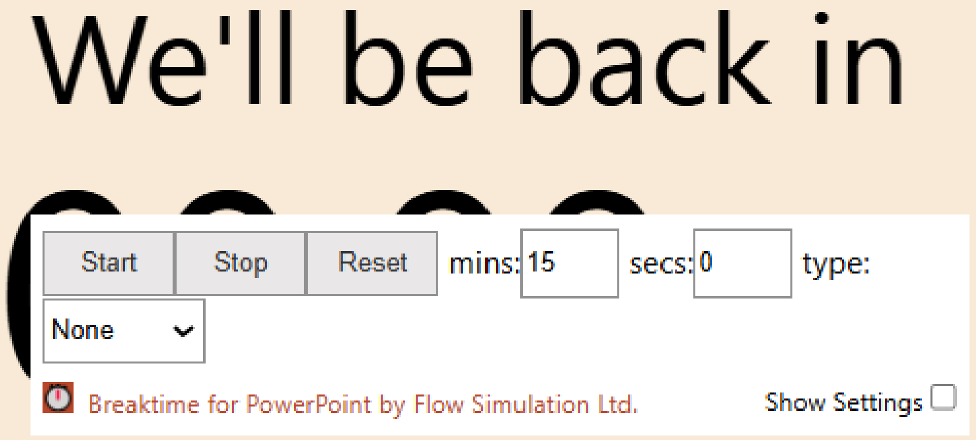
Support for Context Managers

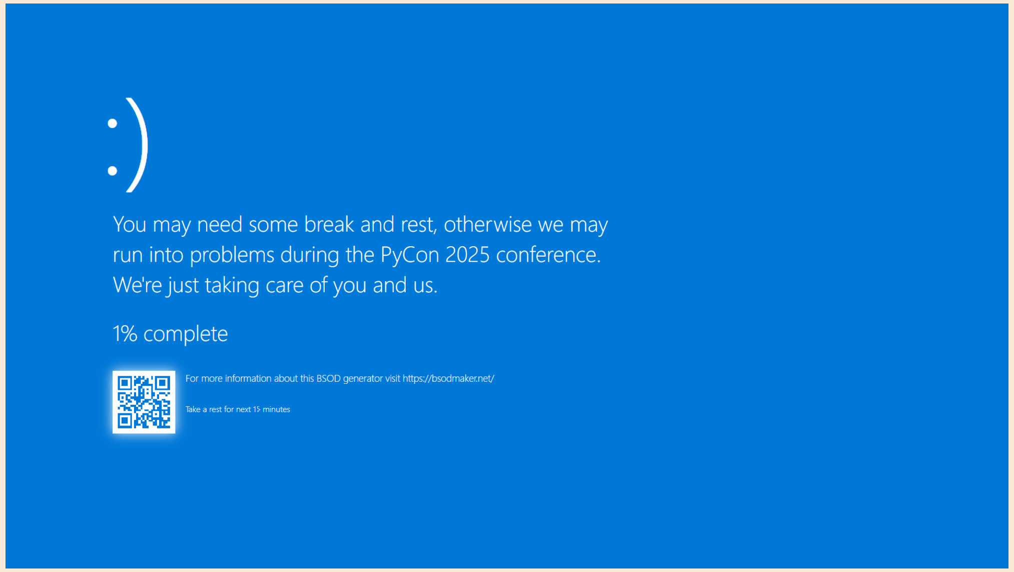
Method	Responsibility
`object.__enter__(self)`	Enters the runtime context related to this object: • sets the runtime context, • obtains resources, • returns an object that can be associated with a variable using the as specifier in the with header

Support for Context Managers

Method	Responsibility
`object.__exit__(self, exc_type, exc_value, traceback)`	Exits the runtime context related to this object: • cleans the runtime context, • releases resources, • handles exceptions

A 15 minute break


We'll be back in


:)

You may need some break and rest, otherwise we may run into problems during the PyCon 2025 conference. We're just taking care of you and us.

1% complete

For more information about this BSOD generator visit https://bsodmaker.net/

Take a rest for next 15 minutes

Try using magic methods

1. Open the `notebooks/tasks.ipynb`
2. `Focus on Task #3`

Support for Iterators

Method	Responsibility
`__iter__(self)`	Initializes the iterator. Returns an interator object
`__next__(self)`	Called to iterate over the iterator. Returns next value or raises the StopIteration exception

Support for containters

Method	Responsibility
`__len__(self)`	Returns the lenght of container
`__getitem__(self, index)`	Returns container element at index/key
`__setitem__(self, index, object)`	Sets value at index/key
`__delitem__(self, index)`	Supports deletion of element
`__contains__(self, object)`	Implements the in operator

Considerations: consistency

Consistency

- Type checking
- Returned object types
- Logging

Considerations: what is returned?

Returned object types

- NotImplemented VS TypeError

Considerations: caching

Impact of Magic Methods on Performance

- complex operatations can significantly lower performance if used often

Strategies

- using __slots__
- caching
- direct access rather additional implicit access

Considerations: documentation

Use `DocStrings` if it’s behavior deviates

Considerations: when to use

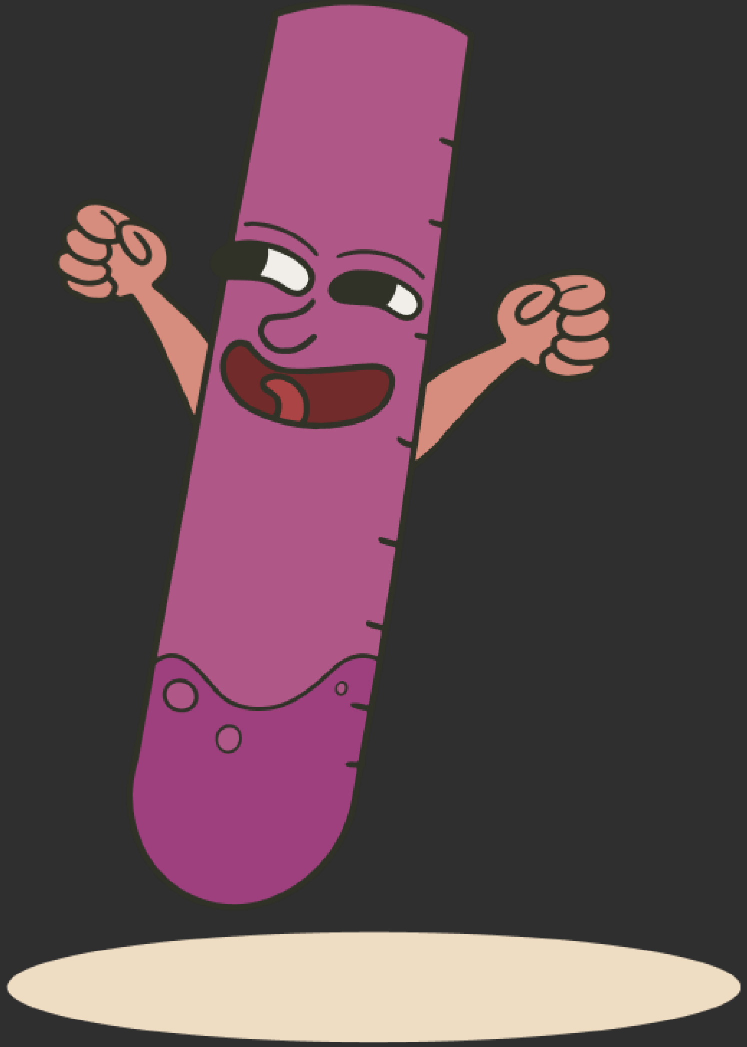
- Create own data structures
- Implement domain-specific types
- Add resources' mangement layer
- Add special behavior to your classes
- Make your code more Pythonic

Considerations: when to avoid

- Simple / built-in attribute access is enough
- Too complex behavior
- Too complex implementation
- Performance is suffering

More information:

https://docs.python.org/3/reference/datamodel.html#specialnames



Thank you for your attention!

Contact info:
pawel.zal@gmail.com
pzal@openedg.org