



# PyConUS 2025

Pittsburgh



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

Paweł Żal, OpenEDG



# Tutorial resources

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

<https://tinyurl.com/PyMagic2025>



# 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 **double underscore** (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 the following Python operators:

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





# Arithmetic operator overloading

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



# 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)
<code>+=</code>	<code>__iadd__(self, other)</code>
<code>-=</code>	<code>__isub__(self, other)</code>
<code>*=</code>	<code>__imul__(self, other)</code>
<code>/=</code>	<code>__itruediv__(self, other)</code>
<code>//=</code>	<code>__ifloordiv__(self, other)</code>
<code>%=</code>	<code>__imod__(self, other)</code>
<code>**=</code>	<code>__ipow__(self, other[, modulo])</code>



# 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)
+	<code>__radd__(self, other)</code>
-	<code>__rsub__(self, other)</code>
*	<code>__rmul__(self, other)</code>
/	<code>__rtruediv__(self, other)</code>
//	<code>__rfloordiv__(self, other)</code>
%	<code>__rmod__(self, other)</code>
**	<code>__rpow__(self, other[, modulo])</code>



# Unary operator overloading

Operator	Magic Method
+	<code>__pos__(self)</code>
-	<code>__neg__(self)</code>
<code>abs()</code>	<code>__abs__(self)</code>
~	<code>__invert__(self)</code>



# Comparison operator overloading

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





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

Start Stop Reset 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 access limiting, logging, data enriching etc.



# Object introspection

Open the notebooks/notebook02.ipynb



# Object introspection

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



# Object lifecycle and customization

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



# Controlling attribute access

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



# Making the object callable

Method	Responsibility
<code>__call__(self, *args, **kwargs)</code>	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
<code>object.__enter__(self)</code>	Enters the runtime context related to this object: <ul style="list-style-type: none"><li>• sets the runtime context,</li><li>• obtains resources,</li><li>• returns an object that can be associated with a variable using the <code>as</code> specifier in the <code>with</code> header</li></ul>



# Support for Context Managers

Method	Responsibility
<code>object.__exit__(self, exc_type, exc_value, traceback)</code>	Exits the runtime context related to this object: <ul style="list-style-type: none"><li>• cleans the runtime context,</li><li>• releases resources,</li><li>• handles exceptions</li></ul>




# A 15 minute break

We'll be back in

Start Stop Reset 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 #3



# Support for Iterators

Method	Responsibility
<code>__iter__(self)</code>	Initializes the iterator. Returns an iterator object
<code>__next__(self)</code>	Called to iterate over the iterator. Returns next value or raises the <a href="#">StopIteration</a> exception



# Support for containers

Method	Responsibility
<code>__len__(self)</code>	Returns the length of container
<code>__getitem__(self, index)</code>	Returns container element at index/key
<code>__setitem__(self, index, object)</code>	Sets value at index/key
<code>__delitem__(self, index)</code>	Supports deletion of element
<code>__contains__(self, object)</code>	Implements the <code>in</code> 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 operations can significantly lower performance if used often

## Strategies

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



# Considerations: documentation

Use DocStrings if the behavior of the method deviates from the norm.



# Considerations: when to use

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



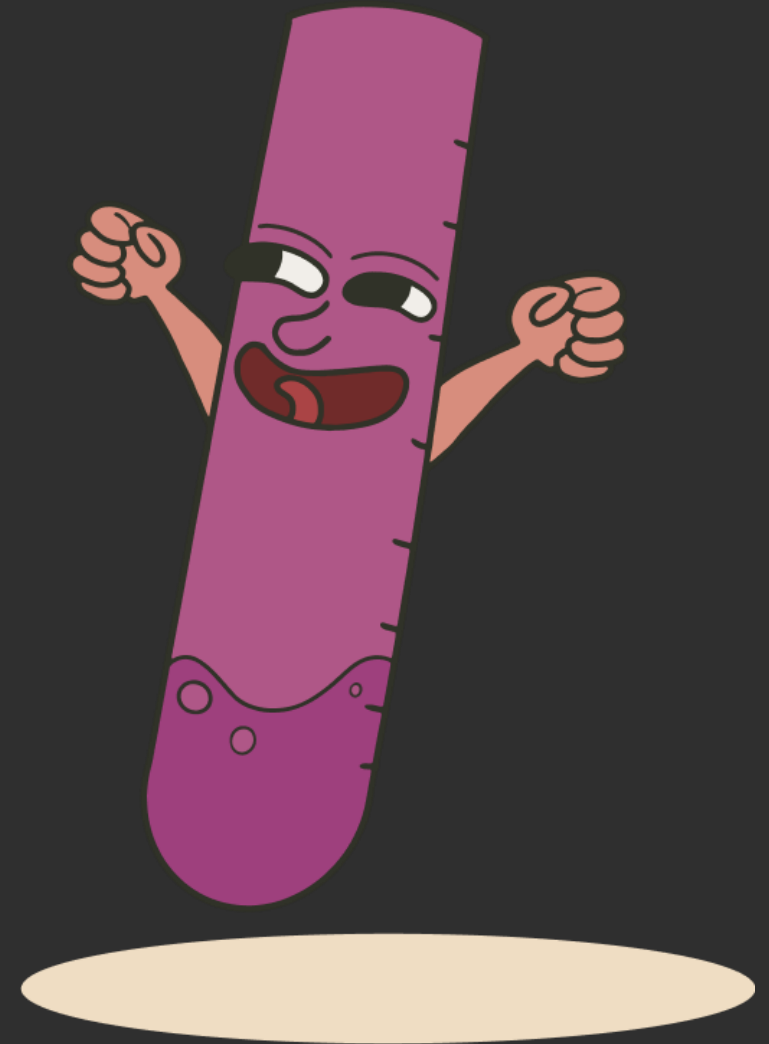
# Considerations: when to avoid

- 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](mailto:pawel.zal@gmail.com)

[pzal@openedg.org](mailto:pzal@openedg.org)