



Lecture 9 – Operator Overloading

CS2513

Cathal Hoare

**A TRADITION OF
INDEPENDENT
THINKING**



UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Lecture Contents

Inheritance

Operator Overloading

- The result of the '*' operator depended on the types passed to it.
- Python (and some other OO languages) allow us to define the meaning operations built into Python when applied to our classes (for example - +, *, <, ==, !=, etc)

Operator Overloading



- But just because we can, doesn't mean we should!
- Before implementing these features, we should consider:
 - Do we need them?
 - And is the overloading appropriate?

Operator Overloading

- For Example:

```
john = Person("John Smith", "Engineer", 55000)
mary = Person("Mary Smith", "Engineer", 65000)
if mary == john:
    print("These both work at the same job")
```

- Does such a comparison make sense? Would it not be more appropriate to have a method `haveSameJob()`
- On the other hand, if the Person class included a unique ID such as a social security number then we could reasonably argue that there is a mechanism to check if two instances were equivalent.

Operator Overloading

- Python provides a large list of built in operators that can be overloaded.
- It does this by providing a method header that can be implemented in a class:
- For instance, `__eq__(self, other_person)` is called when we compare two person objects for equivalence.
- We must provide an implementation for this in our Person class if we wish to support the `'=='` operator.

Operator Overloading



- We have seen some of these already:
 - `__init__`
 - `__str__`
- They are recognised by the double underscore and are called ‘magic’ methods...because they add a little ‘magic’ to your class.
- They are also known as ‘Dunder’ methods.
- We can inspect a type’s (or class) methods using `dir()`

Operator Overloading

- `__add__` - allows us to add (+) two instances together
- `__lt__` - allows us to check if one object is less than another
- `__eq__` - allows us to check if one object is equivalent to another
- `__ne__` - allows us to check if one object is not equivalent to another



Next Time:
Review Examples