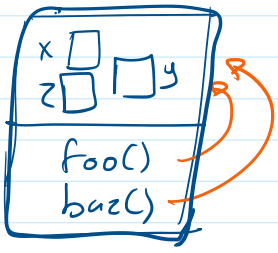
Ch9 Classes

Friday, November 10, 2023 8:37 AM

Object Oriented Programming.

A form of code organization that couples together data (variables) and operations (functions)

Memory



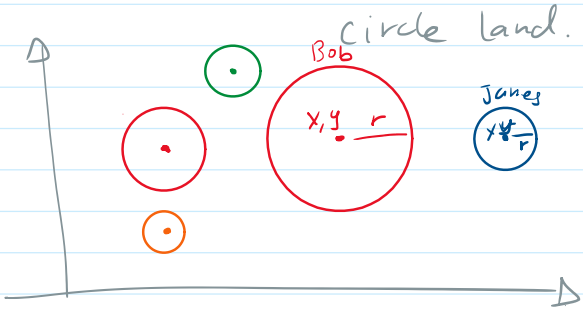
'80
- Simula
- Smalltalk by Alan Key.

Pillars
OOP
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Python - Abstraction + some Inheritance

Motivating Example:

Circles



the OOP approach is create(declare) a new "type" to represent what we want.

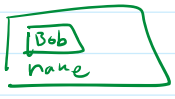
↑ class.

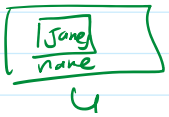
Syntax `class classname :`

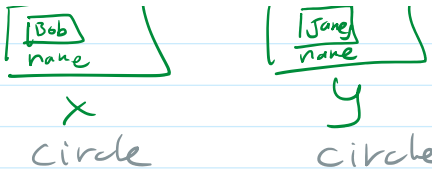
} variable and function declaration.

E.g.

Memory





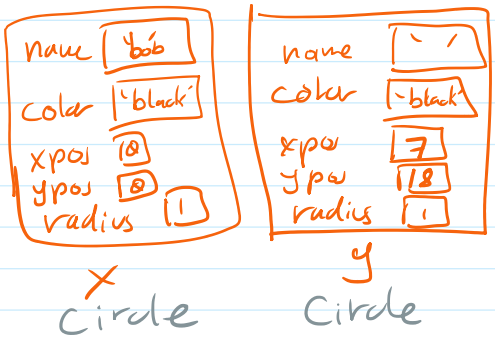


- "Constructor"

  `__init__(self)`

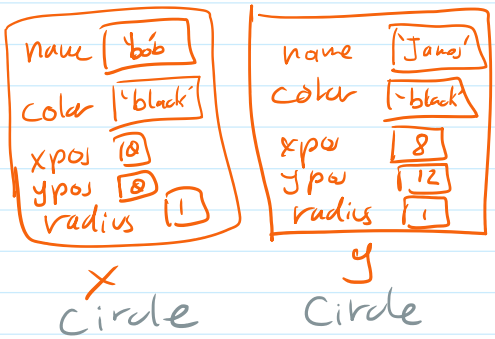
`x = Circle()`

Memory



Constructors can also have parameters, (use default values)

Memory



- Class attributes.
  - A member variable shared by all objects of the same class (instances).
  - declared directly in the class.
  - usually intended for values that should not be changed.

- Class "interface"
  - the collection of member variables ("fields") and member functions ("methods") expected in an object of a particular class.

expected in an object of a particular class.

- Introspection:

  You can check for a variable's class with
  `isinstance(var, type)`

- Special Methods.

  `__str__()` Print variables. should return a string.

  operator overloading:

  `__lt__()` implement < operator.

  e.g bob < alice.

  ↓

  `bob.__lt__(alice)`

  Other operators.

>	`__gt__`
or	`__or__`
and	`__and__`
int	`__int__` Convert to integer
float	`__float__` Convert to float.

  more complex example.

  \+ `__add__`

  bob + alice ↷ return a new Circle between bob and alice.

  bob + 5 ↷ add to bob's radius.

  `bob.__add__()`