9 Classes

Monday, November 11, 2024 8:04 AM

OOP.- Object Oriented Programming.

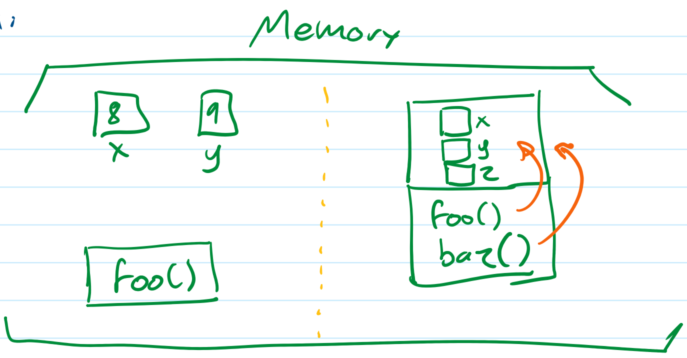
- A form of code organization that couples data (variables) and their operators (functions)

- Terminology varies:

variables = fields = members variables

functions = methods = members functions

- idea:



HISTORY:

'80 Simula

Alan Key Small-Talk

and later

C++, Java, C#,.....

FEATURES:

"Pillars" of OOP

- Abstraction.- the ability to create new types.
- Encapsulation.- Types should keep their data Hidden
- Inheritance.- the ability to define a type as an extension of another type.
- Polymorphism.- A type may behave differently according to its internal circumstances

Python ::

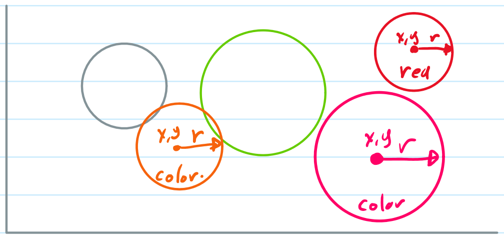
- Abstraction. ✓
- Ecapsulation ⊗
- Inheritance. ok
- Polymorphism ok

- Inheritance. ok
- Polymorphism ok

ILLUSTRATIVE EXAMPLE:

- create new types:- called classes.
- variables of these new types are called objects / instances.

e.g. Circles:



Create a new type to represent "Circles"

variables : $\boxed{\begin{matrix} x \\ y \\ r \\ \text{color.} \end{matrix}}$

Syntax:

class classname :

} functions (variables)

in order to initialize a class, a special function

def __init__ (self, parameters)

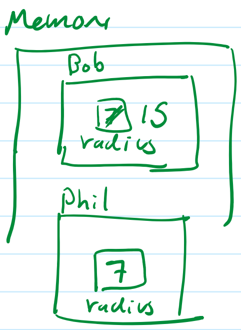
← to provide initial values to the member variables

e.g. sketch 1.

```
class circle :
   def __init__(self) :
      self.radius = 7

bob = circle()

phil = circle()
```

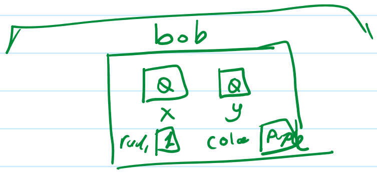
← defines class type "circle"

← creates an object/instance of class circle.

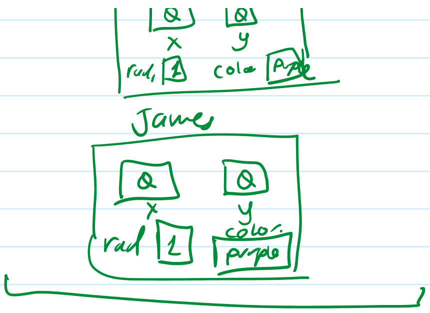


e.g. sketch 2.

```
class circle :
   def __init__(self) :
      self.x = 0
      self.y = 0
      self.radius = 1
```

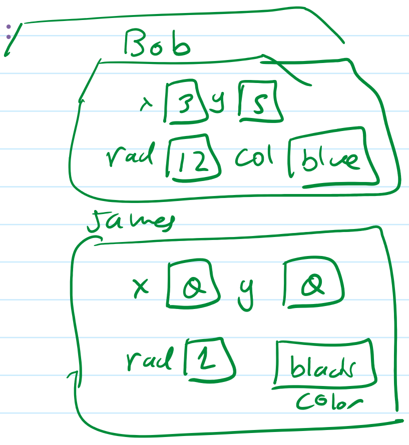


```
            self.y = 0
            self.radius = 1
            self.color = purple

        bob = circle()
        james = circle()
```



e.g sketch 3:

```
    class circle :
       def __init__(self, a=0, b=0, r=1, c='black') :
          self.x = a
          self.y = b
          self.radius = r
          self.color = c

    Bob = circle(XXXX, 3, 5, 12, 'blue')
    James = circle()
```

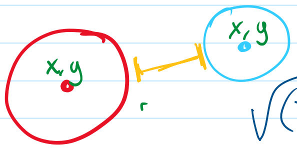


Member Functions / Methods:

get_area()

get_circumference()

distance_to(circle)



$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - r_1 - r_2$$

```
    class circle :
       def __init__(self, a=0, b=0, r=1, c='black') :
           self.x = a
           self.y = b
           self.radius = r
           self.color = c

        def get_area(self) :
           pi = 3.141519
           area = pi * self.radius * self.radius
           return area

        def get_circunference(self) :
           pi = 3.14159
           circ = pi * self.radius
           return circ

        def distance_to(self, circle2) :
           xd = self.x - circle2.x
           yd = self.y - circle2.y
           dist = math.sqrt( xd*xd + yd*yd )
           dist = dist - self.radius - circle2.radius
           return dist
```

- Class Interface :-
  the collection of member variables (fields) member functions (Methods) expected from an object of a particular class.
- Introspection :
  you can ask an object its class.
  isinstance(var, type)
- Special methods :
  __str__() • turn an object into a string
  • print object.

e.g.
(x, y - r : color)

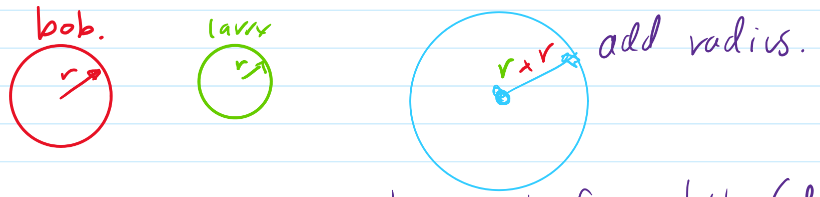
__int__() • turns an object into an integer
__float__() • turns an object into a float.

- Operator oberloading .
  Use objects like mathematical entities.
  e.g.
  bob + james
  bob < james
  bob * 5
  bob == james.

- bob == james
  __eq__()
- bob < james
  < __lt__()
  > __gt__()
  ≤ __le__()
  ≥ __ge__()
-

- +

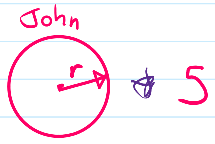
bob + larry



center point from bob (lhs)
Color from larry (rhs)

__add__() bob + larry
$\Updownarrow$
bob.__add__(larry)

- *



__mul__()

return a list of 5 Johns...

Use operator overloading with care.

- SAMPLE CODE

```
import math

class circle :
    def __init__(self, a=0, b=0, r=1, c='black') :
        self.x = a
        self.y = b
        self.radius = r
        self.color = c

    def get_area(self) :
        pi = 3.141519
        area = pi * self.radius * self.radius
        return area

    def get_circunference(self) :
        pi = 3.14159
        circ = pi * self.radius
        return circ

    def distance_to(self, circle2) :
        xd = self.x - circle2.x
        yd = self.y - circle2.y
        dist = math.sqrt( xd*xd + yd*yd )
        dist = dist - self.radius - circle2.radius
        return dist
    def __str__(self) :
        return f'({self.x},{self.y}-{self.radius}:{self.color})'

    def __eq__(self, circle2) :
        return self.radius == circle2.radius

    def __lt__(self, circle2) :
        return self.radius < circle2.radius

    def __add__(self, circle2) :
        circle3 = circle()
        circle3.x = self.x
        circle3.y = self.y
        circle3.color = circle2.color
        circle3.radius = self.radius + circle2.radius
        return circle3

    def copy(self) :
        c = circle()
        c.x = self.x
        c.y = self.y
        c.color = self.color
        c.radius = self.radius
        return c

    def __mul__(self, n) :
        l = []
        for i in range(n) :
            l.append( self.copy() )
        return l
```

EOF