

Inheritance and Polymorphism

Extended Class (Subclass):

Shifted_Circle()

```
class Shifted_Circle(Circle):
```

```
    def __init__(self, x = 0, y = 0, r = 1):
```

```
        Circle.__init__(self, r)
```

```
        self.__x = x
```

```
        self.__y = y
```

```
        self.__radius = r
```

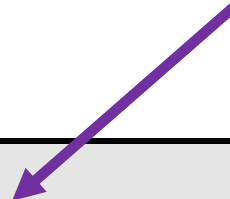
```
    def distance(self):
```

```
        return (self.__x**2 + self.__y**2)**0.5
```

Python Subclass Template

subclass

superclass



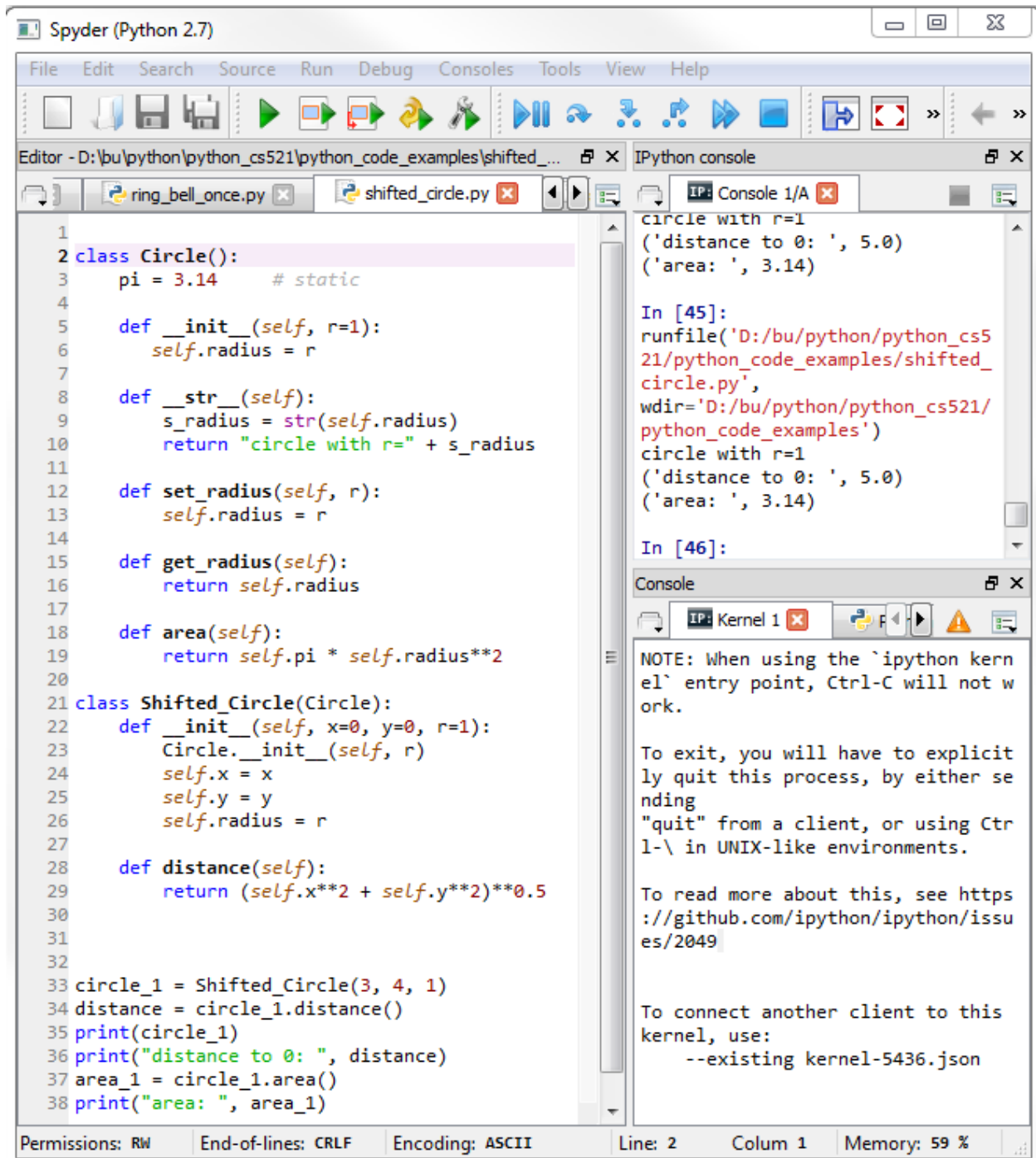
```
Class Shifted_Circle(Circle):
```

```
# new static fields (optional)
```

```
def __init__():          # (new) constructor
```

```
def distance():         # new method(s)
```

Shifted_Circle() Class Run



The screenshot displays the Spyder Python IDE interface. The left pane shows the source code for `shifted_circle.py`. The code defines a `Circle` class with attributes `pi` and `radius`, and methods `__init__`, `__str__`, `set_radius`, `get_radius`, and `area`. A `Shifted_Circle` class is defined as a subclass of `Circle`, inheriting its methods and adding `__init__` and `distance` methods. The `distance` method calculates the distance from the origin (0,0) to the point (x,y) using the formula $\sqrt{x^2 + y^2}$. The right pane shows the IPython console with the output of the code execution. The output shows the creation of a `Shifted_Circle` object with `x=3`, `y=4`, and `r=1`, and the calculation of its distance and area.

```
1 class Circle():
2     pi = 3.14 # static
3
4     def __init__(self, r=1):
5         self.radius = r
6
7     def __str__(self):
8         s_radius = str(self.radius)
9         return "circle with r=" + s_radius
10
11     def set_radius(self, r):
12         self.radius = r
13
14     def get_radius(self):
15         return self.radius
16
17     def area(self):
18         return self.pi * self.radius**2
19
20 class Shifted_Circle(Circle):
21     def __init__(self, x=0, y=0, r=1):
22         Circle.__init__(self, r)
23         self.x = x
24         self.y = y
25         self.radius = r
26
27     def distance(self):
28         return (self.x**2 + self.y**2)**0.5
29
30 circle_1 = Shifted_Circle(3, 4, 1)
31 distance = circle_1.distance()
32 print(circle_1)
33 print("distance to 0: ", distance)
34 area_1 = circle_1.area()
35 print("area: ", area_1)
```

circle with r=1
(distance to 0: ', 5.0)
(area: ', 3.14)

In [45]:
runfile('D:/bu/python/python_cs521/python_code_examples/shifted_circle.py',
wdir='D:/bu/python/python_cs521/python_code_examples')
circle with r=1
(distance to 0: ', 5.0)
(area: ', 3.14)

In [46]:

Console

IP: Kernel 1

NOTE: When using the `ipython kernel` entry point, Ctrl-C will not work.

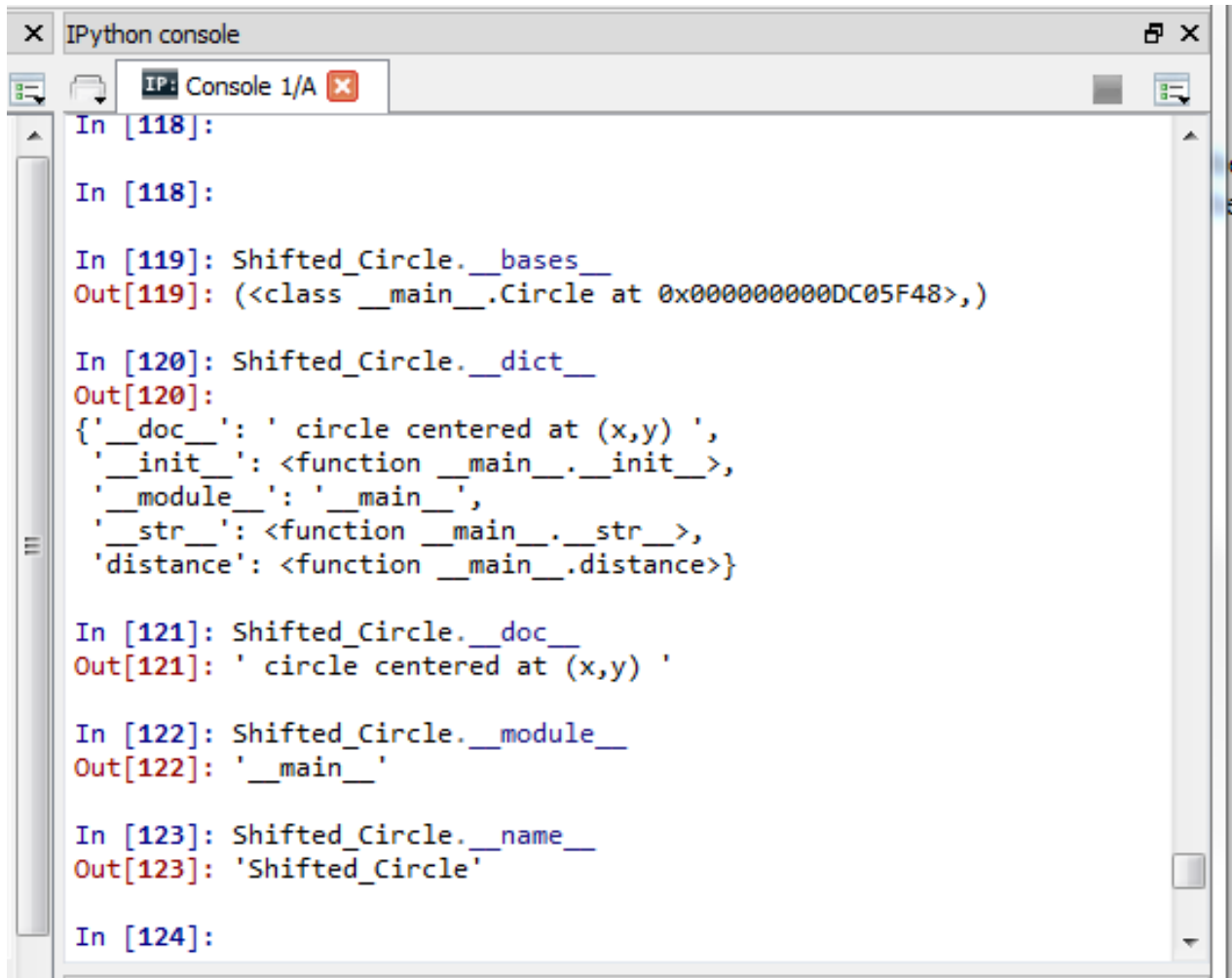
To exit, you will have to explicitly quit this process, by either sending "quit" from a client, or using Ctrl-\ in UNIX-like environments.

To read more about this, see <https://github.com/ipython/ipython/issues/2049>

To connect another client to this kernel, use:
--existing kernel-5436.json

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 2 Column 1 Memory: 59 %

Shifted_Circle() Attributes



```
IPython console
IP: Console 1/A x

In [118]:

In [118]:

In [119]: Shifted_Circle.__bases__
Out[119]: (<class __main__.Circle at 0x000000000DC05F48>,)

In [120]: Shifted_Circle.__dict__
Out[120]:
{'__doc__': ' circle centered at (x,y) ',
 '__init__': <function __main__.__init__>,
 '__module__': '__main__',
 '__str__': <function __main__.__str__>,
 'distance': <function __main__.distance>}

In [121]: Shifted_Circle.__doc__
Out[121]: ' circle centered at (x,y) '

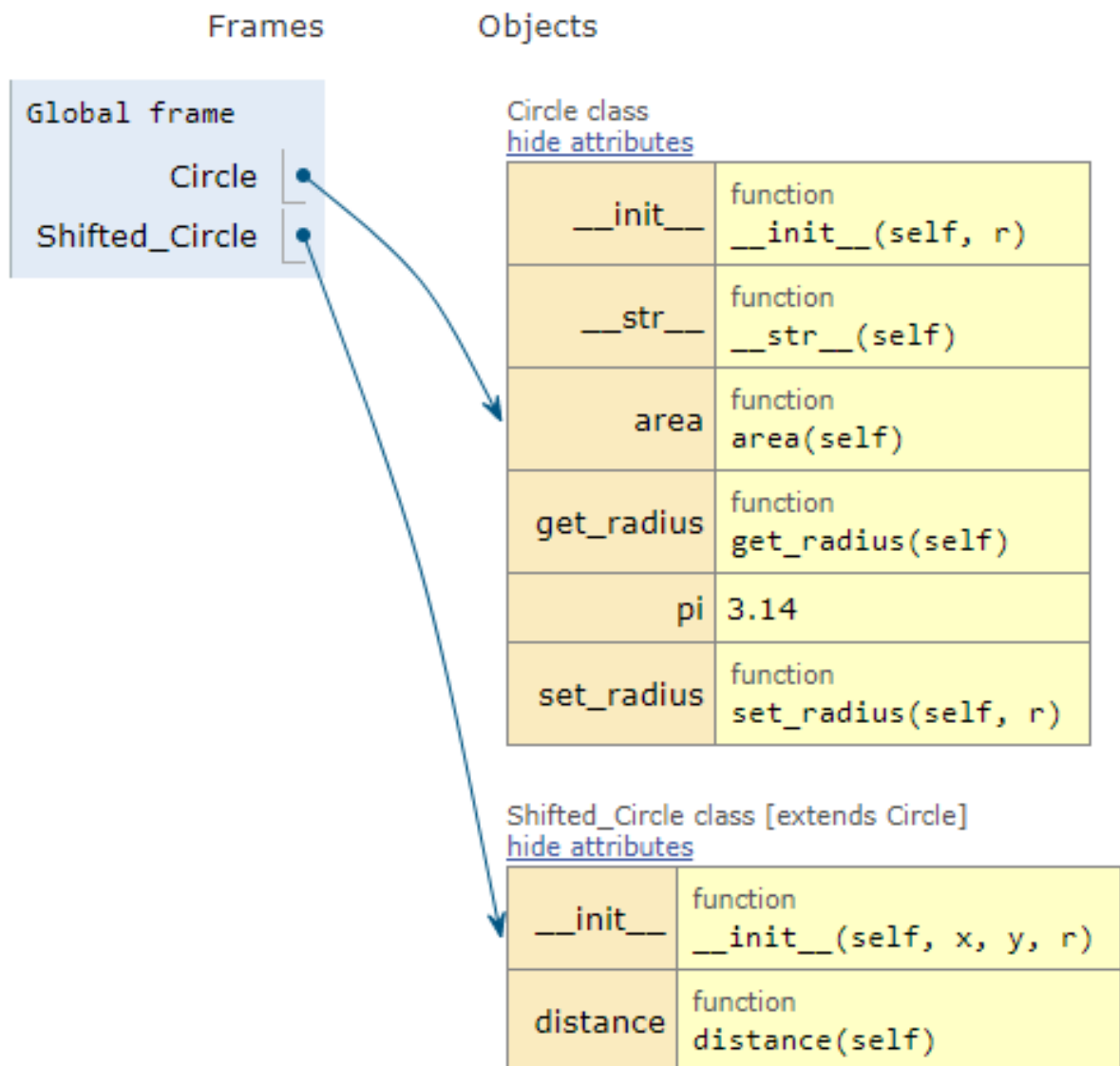
In [122]: Shifted_Circle.__module__
Out[122]: '__main__'

In [123]: Shifted_Circle.__name__
Out[123]: 'Shifted_Circle'

In [124]:
```

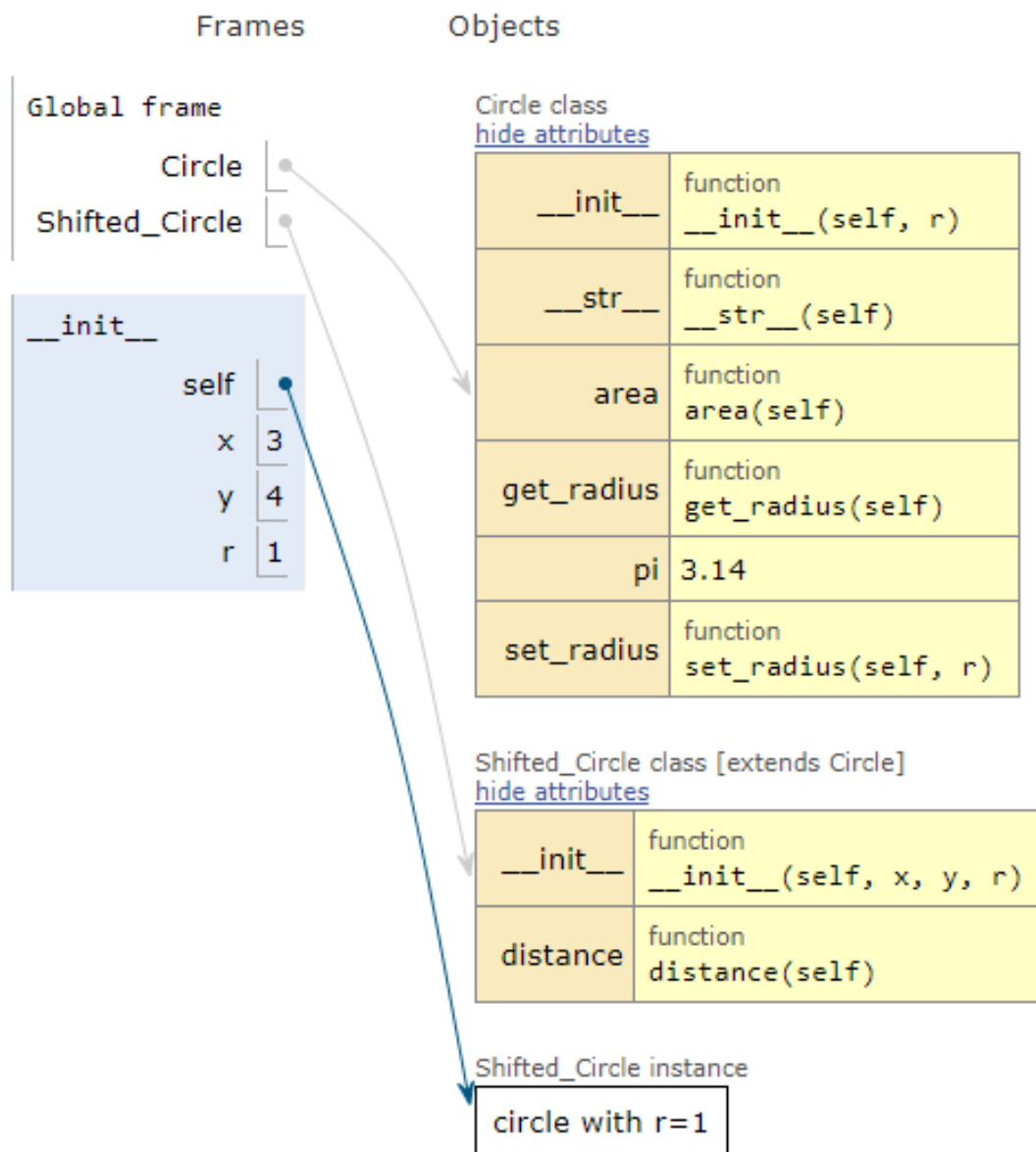
Shifted_Circle() Class

```
>>> import shifted_circle
```



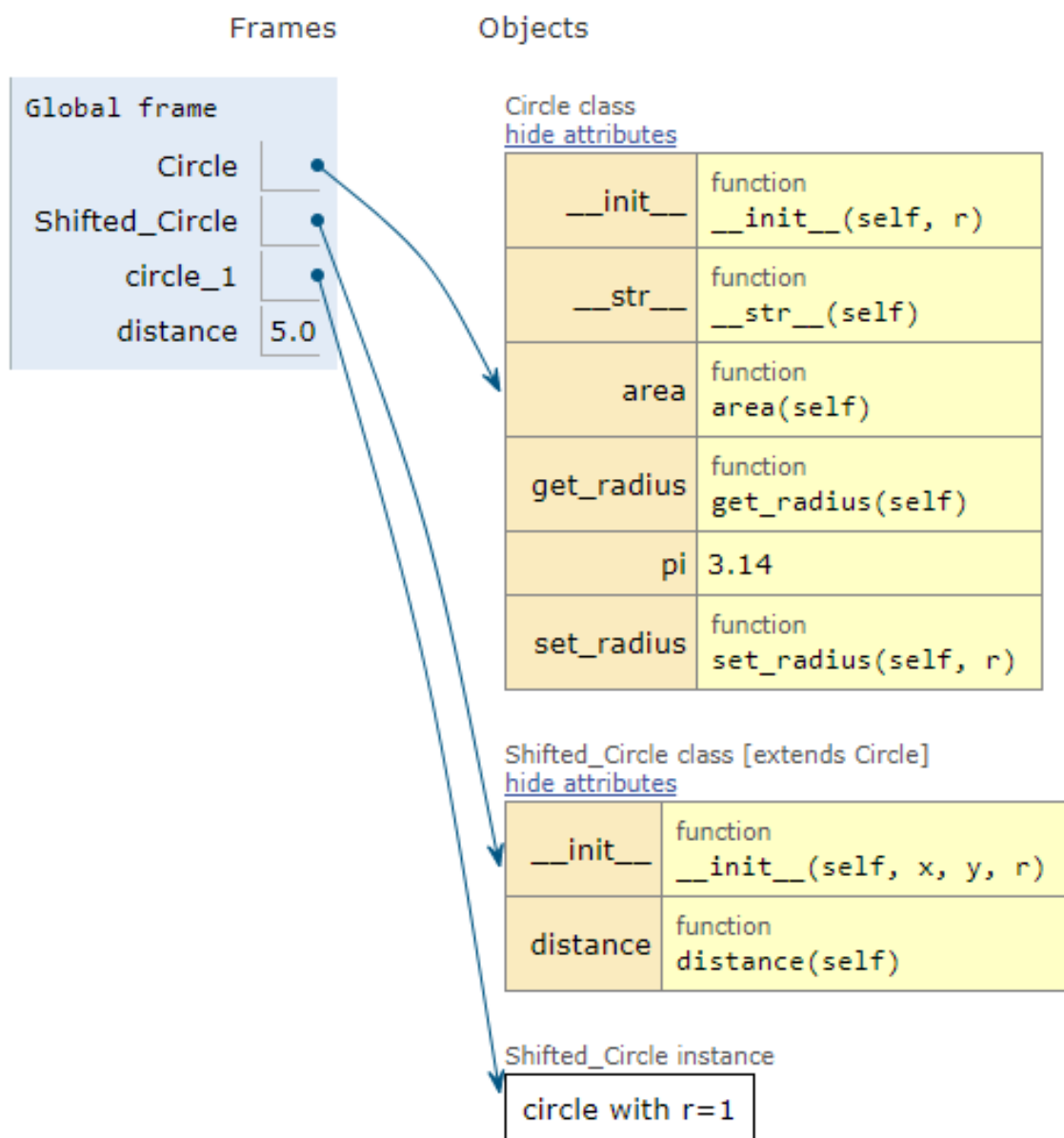
Shifted_Circle() Class (cont'd)

```
>>> circle_1 = Shifted_Circle(3, 4, 1)
```



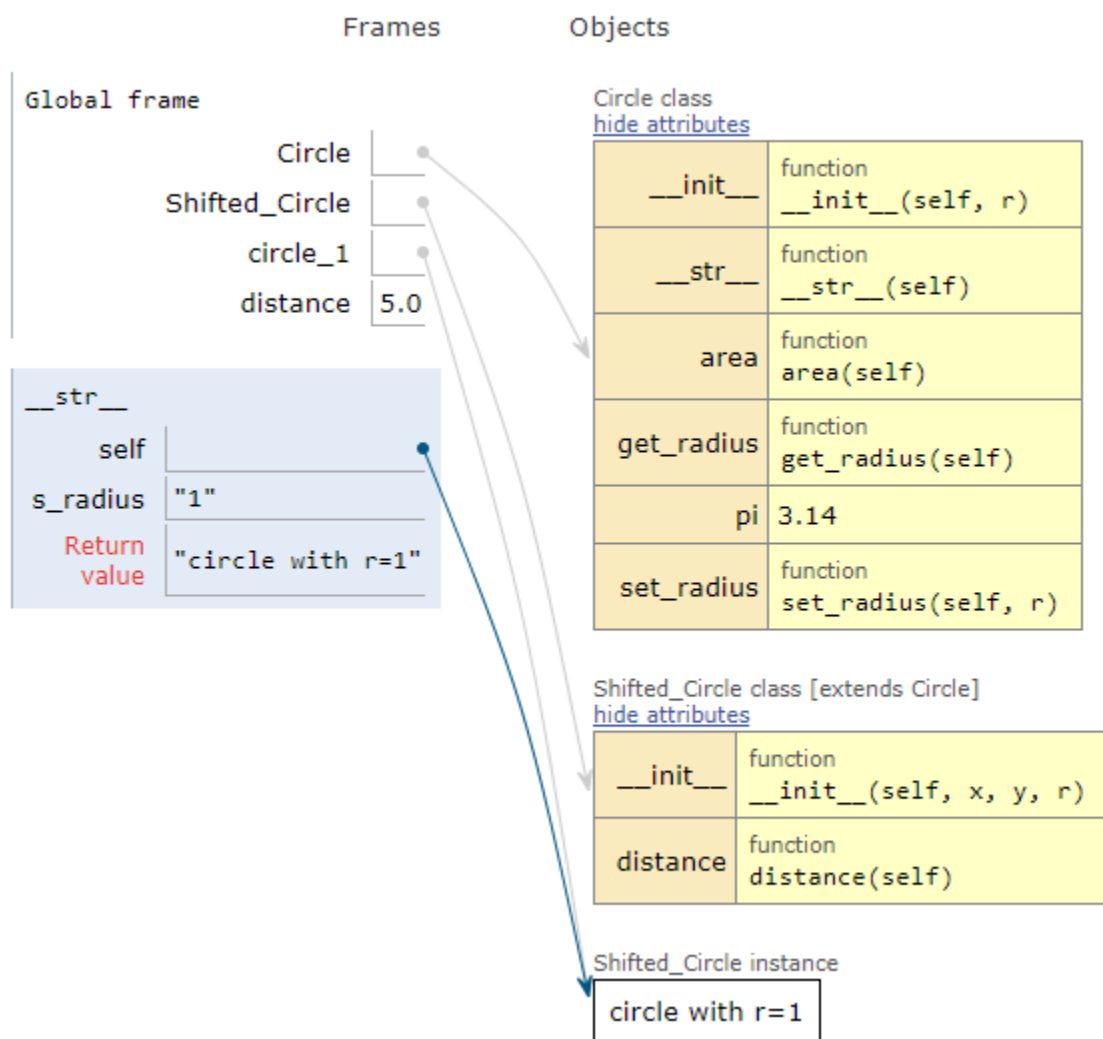
Shifted_Circle() Class (cont'd)

```
>>> distance = circle_1.get_distance()
```



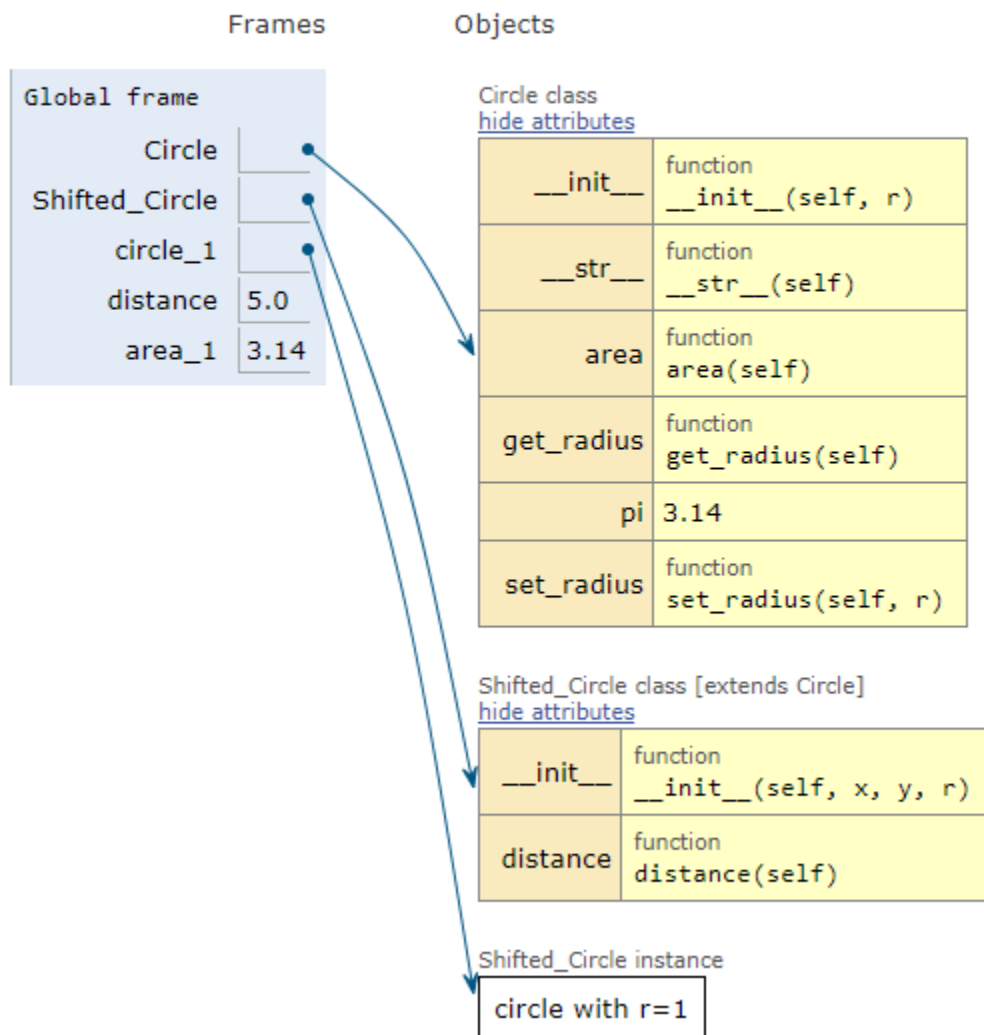
Shifted_Circle() Class (cont'd)

```
>>> print(circle_1)
```



Shifted_Circle() Class (cont'd)

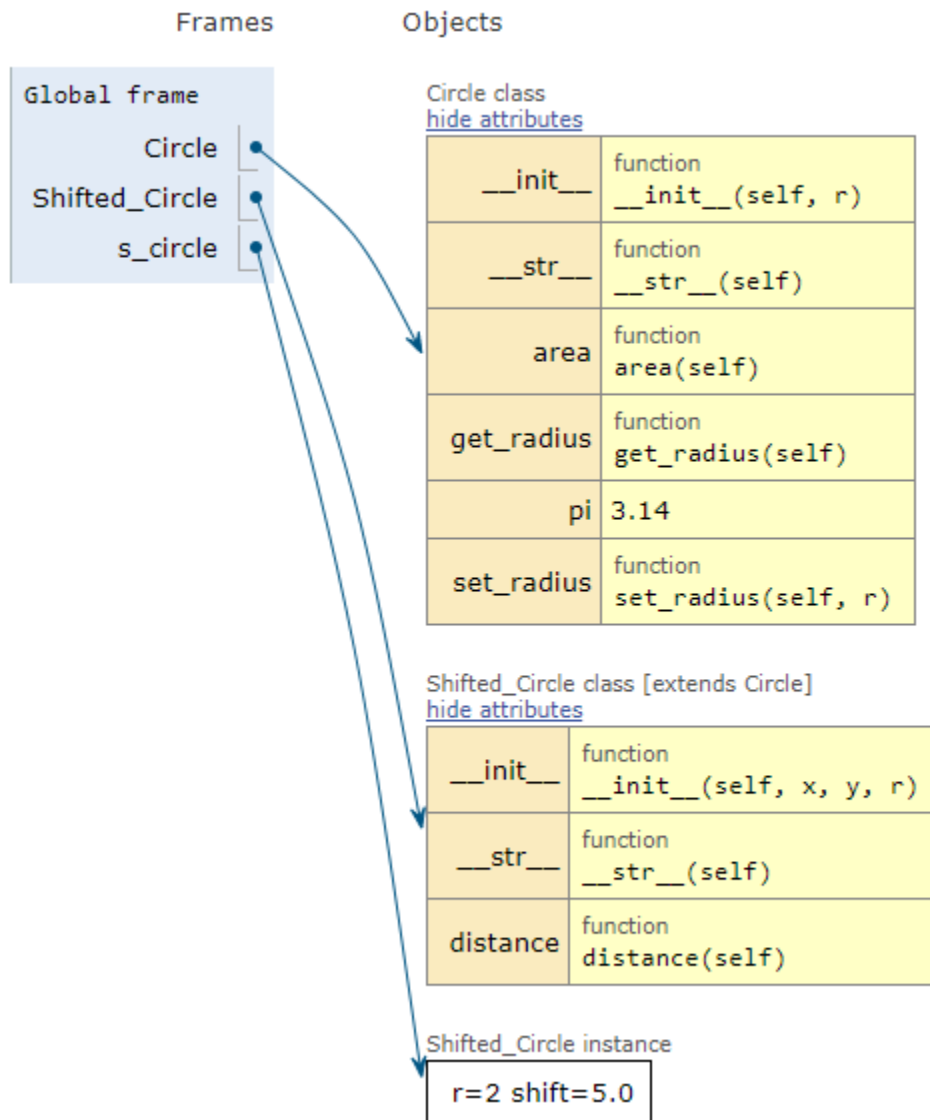
```
>>> area_1 = circle_1.area()
```



Inheritance

- extended class inherits *area()* method

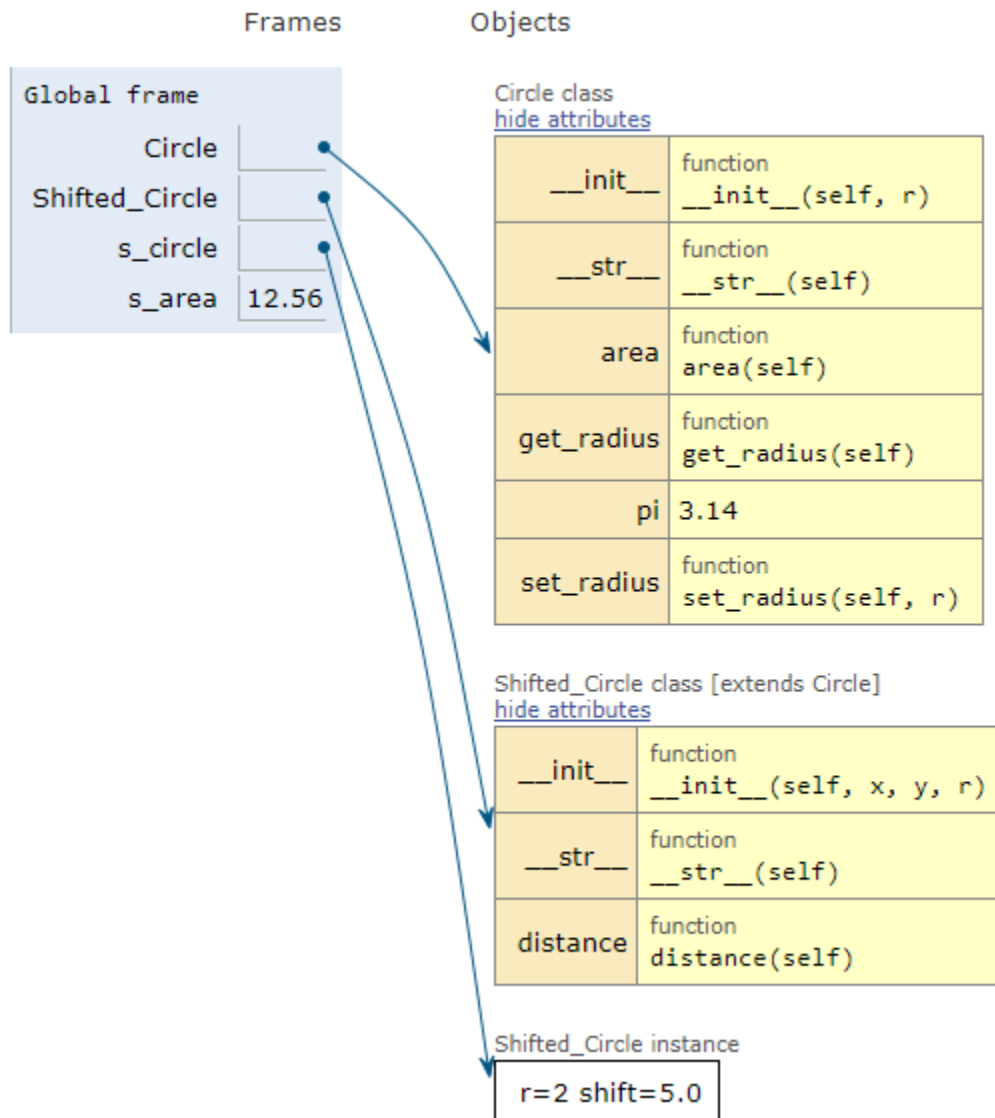
```
>>> s_circle = Shifted_Circle(3, 4, 2)
```



Inheritance (cont'd)

- extended class inherits *area()* method

```
>>> s_area = s_circle.area()
```



Overloading

- same method with different signatures
- `set_radius(r)` and `set_radius(x, y, r)` in extended class

```
>>> s_circle = Shifted_Circle(3, 4, 2)
>>> print(s_circle)
>>>
>>> # set_radius()
>>> s_circle.set_radius(5)
>>> print(s_circle)
>>>
>>> # set_radius()
>>> s_circle.set_radius(10, 12, 20)
>>> print(s_circle)
```

Overloading *set_radius()*

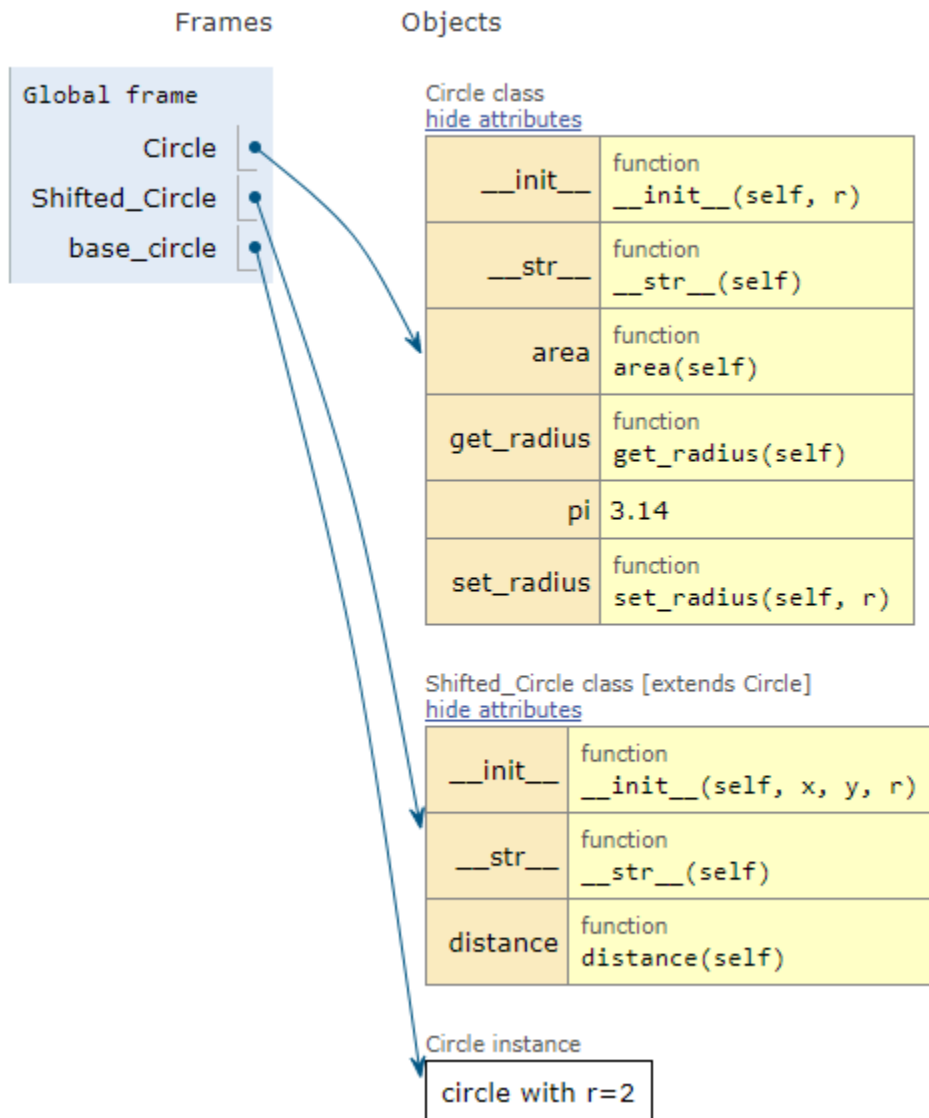
```
class Shifted_Circle(Circle):  
    def __init__(self, x = 0, y = 0, r = 1):  
        Circle.__init__(self, r)  
        self.x = x  
        self.y = y  
        self.radius = r  
    def distance(self):  
        return (self.x**2 + self.y**2)**0.5  
    def set_radius(self, x, y, r):  
        self.x = x  
        self.y = y  
        self.radius = r  
    def __str__(self):                # representation.  
        s_radius = str(self.radius)  # local variable  
        s_distance = str(self.distance())  
        return " r=" + s_radius + " shift=" + s_distance
```

Overriding

- override method(s) in base class

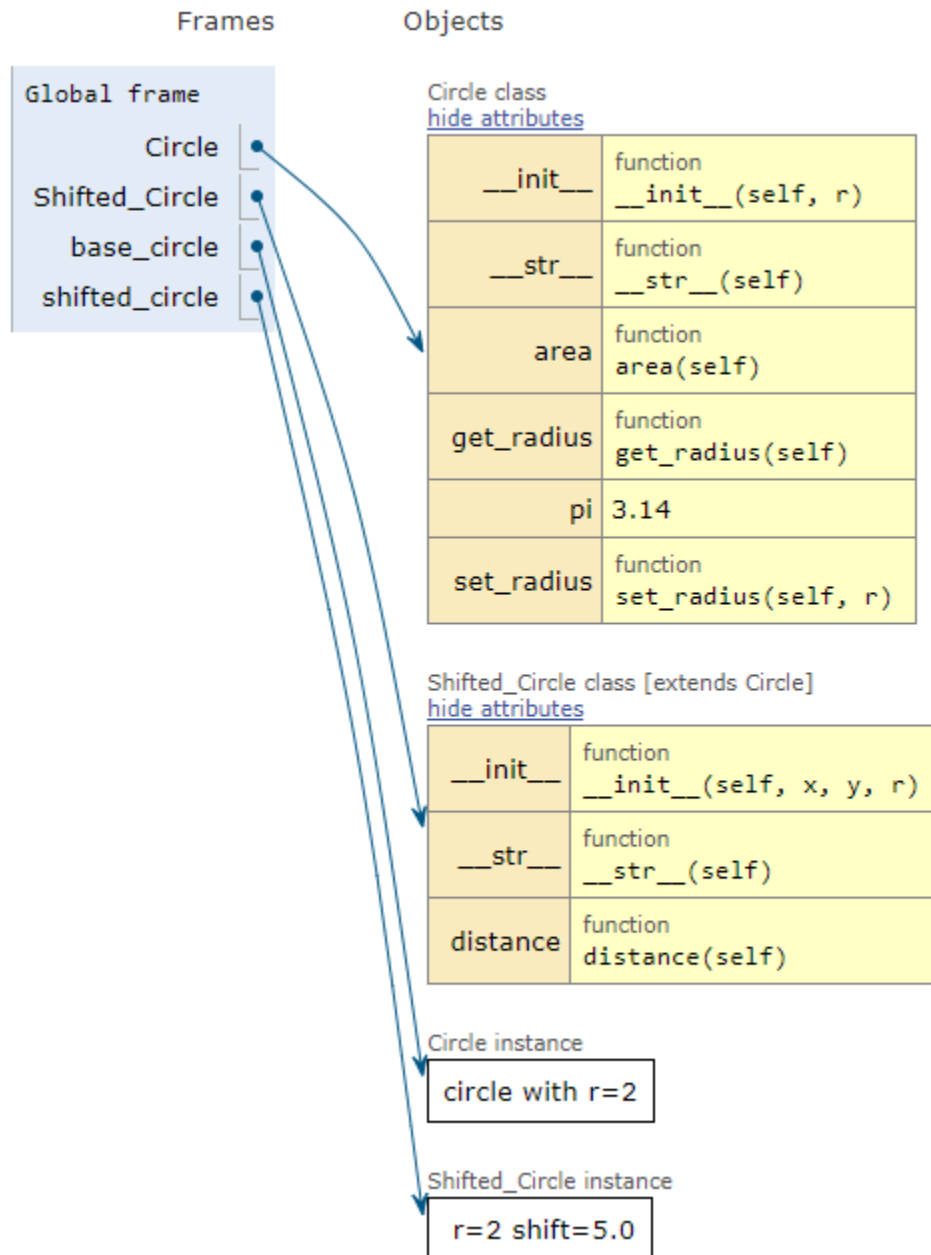
```
>>> base_circle = Circle(2)
```

```
>>> print(base_circle)
```



Overriding (cont'd)

```
>>> shifted_circle = Shifted_Circle(3,4,2)
>>> print(shifted_circle)
```



Overriding `__str__()` Method

```
class Shifted_Circle(Circle):  
    def __init__(self, x = 0, y = 0, r = 1):  
        Circle.__init__(self, r)  
        self.x = x  
        self.y = y  
        self.radius = r  
  
    def distance(self):  
        return (self.x**2 + self.y**2)**0.5
```

```
def __str__(self):  
    s_radius = str(self.radius)  
    s_distance = str(self.distance())  
    return " r=" + s_radius + " shift=" + s_distance
```

Polymorphism

- *polys* (“many”), *morphe* (“form”)
- same interface for different types

	0	1	2	3	4	5	6	7	8	9	
['M',	'o',	'z',	'z',	'a',	'r',	'e',	'l',	'l',	'a']
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

```
>>> y = ['M','o','z','z','a','r','e','l','l','a'][2]
```

	0	1	2	3	4	5	6	7	8	9	
('M',	'o',	'z',	'z',	'a',	'r',	'e',	'l',	'l',	'a')
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

```
>>> y = ('M','o','z','z','a','r','e','l','l','a')[2]
```

	0	1	2	3	4	5	6	7	8	9
	M	o	z	z	a	r	e	l	l	a
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> y = "Mozzarella"[2]
```

Polymorphism Example

```
class Animal():  
    def __init__(self, name):    self.name = name  
    def talk(self):    raise NotImplementedError("left to subclass")
```

```
class Cat(Animal):  
    def talk(self):    return "Meou!"
```

```
class Dog(Animal):  
    def talk(self):    return "Woof! Woof!"
```

```
>>> from Animal import Cat, Dog  
>>> animals=[Cat('Missy'),Cat('Smokey'),Dog('Max')]  
>>> for animal in animals:  
        print animal.name + ':' + animal.talk()  
>>> Missy: Meow!  
>>> Smokey: Meow!  
>>> Max: Woof! Woof!
```

Summary of Object-Oriented Concepts

- classes implement abstraction
- encapsulation: details are hidden
- applications use class methods
- subclasses inherit or override superclass methods
- polymorphism: same interface for different types

Review Problems

Interview Problem

- difference between overloading and overriding

Interview Problem

- explain inheritance with an example

Interview Problem

- which methods of Python are used to determine the type of instance and inheritance?

Interview Problem

- consider the case of multiple inheritance: a child class C is derived from two base classes say A and B as:
class C(A, B)
- which parent class's method will be invoked by the interpreter whenever object of class C calls a method func() that is existing in both the parent classes say A and B and does not exist in class C?

Interview Problem

- how are *inheritance* and *overriding* methods related?