

# Python

# Classes

# Procedural vs. Object Oriented Programming

- procedural programming:
  1. functions operate on data
  2. data is passed between procedures
- object-oriented programming:
  1. encapsulation: functions and data are combined into object
  2. objects provide *methods* to operate on *data attributes*
- outside programs do not know the internal details of an object

# Base Class: *Circle()*

```
class Circle():
    pi = 3.14                                # static variable
    def __init__(self, r = 1):                  # constructor
        self.radius = r                         # with default

    def __str__(self):                          # representation
        s_radius = str(self.radius)            # local variable
        return "circle r=" + s_radius

    def set_radius(self, r):                   # mutator
        self.radius = r                        # (setter)

    def get_radius(self):                      # accessor
        return self.radius                     # (getter)

    def area(self):                           # method
        return self.pi * self.radius**2
```

# Python Class Template

```
Class Circle()
```

```
# class name
```

```
pi = 3.14
```

```
# (static) data fields
```

```
# Methods  
def __init__()  
# constructor  
def __str__()  
# representation  
def set_radius()  
# setter  
def get_radius()  
# getter  
def area()  
# other methods
```

- class methods use “dot” notation

```
>>> circle = Circle(5)  
>>> area = circle.area()
```

# Base *Circle()* Class Run

The screenshot shows the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Tools, View, and Help. Below the menu is a toolbar with various icons for file operations and code execution. The main area is divided into three panes: an Editor pane containing Python code, an IPython console pane, and a Console pane.

**Editor:** Displays the following Python code:

```
1 # all variables are public
2 class Circle():
3     pi = 3.14      # static
4
5     def __init__(self, r=1):
6         self.radius = r
7
8     def __str__(self):
9         s_radius = str(self.radius)
10        return "circle with r=" + s_radius
11
12    def set_radius(self, r):
13        self.radius = r
14
15    def get_radius(self):
16        return self.radius
17
18    def area(self):
19        return self.pi * self.radius**2
20
21 circle = Circle()
22 print(circle)
23 area = circle.area()
24 print("area: ", area)
```

**IPython console:** Shows the following session:

```
In [61]:
In [61]:
In [62]:
runfile('D:/bu/python/python_cs521/python_code_examples/circle.py',
        wdir='D:/bu/python/python_cs521/python_code_examples')
circle with r=1
('area: ', 3.14)

In [63]:
```

**Console:** Displays a note about exiting the kernel:

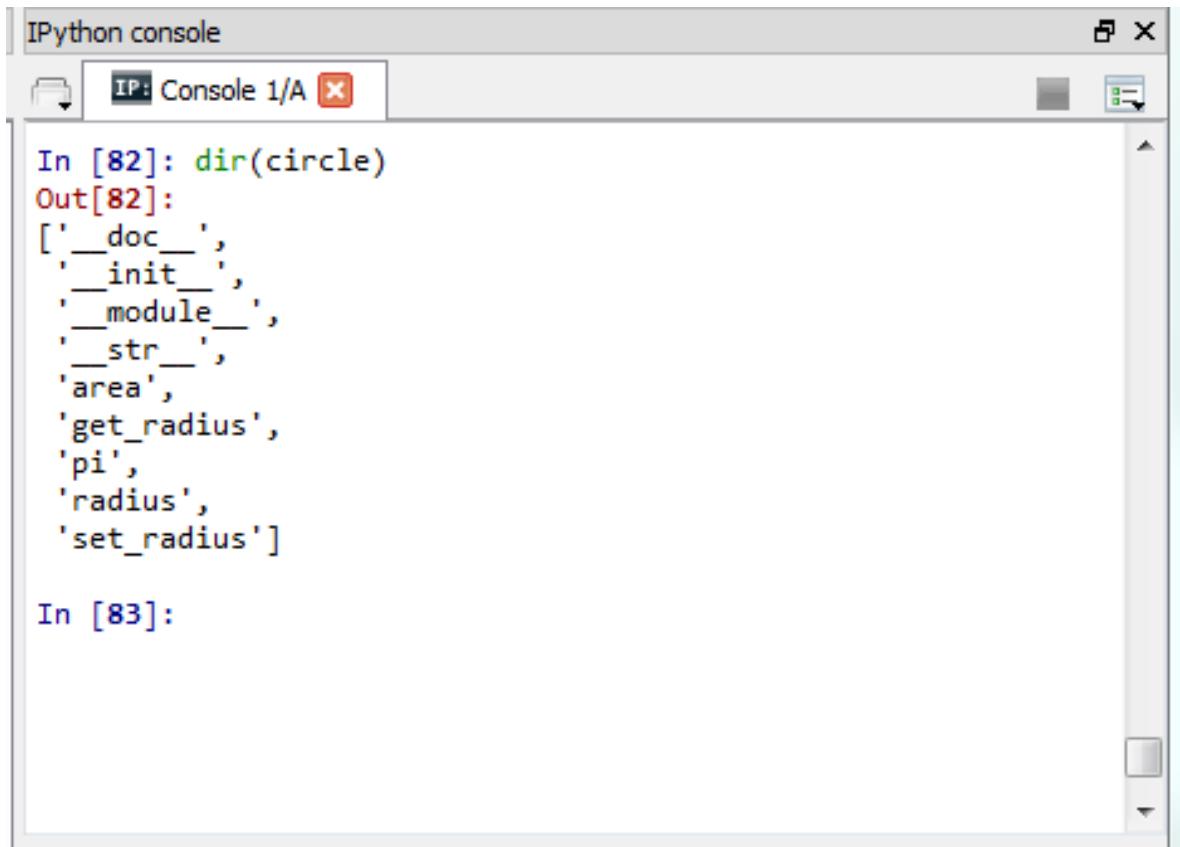
```
NOTE: When using the `ipython kernel` entry point, Ctrl-C will not work.
```

At the bottom, status information is shown: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 21, Colur 17, Memory: 51 %.

# Class Methods and Variables

- use `dir()` command

```
>>> dir(circle)
```



The screenshot shows an IPython console window titled "IPython console" with a tab labeled "IP: Console 1/A". The console displays the following interaction:

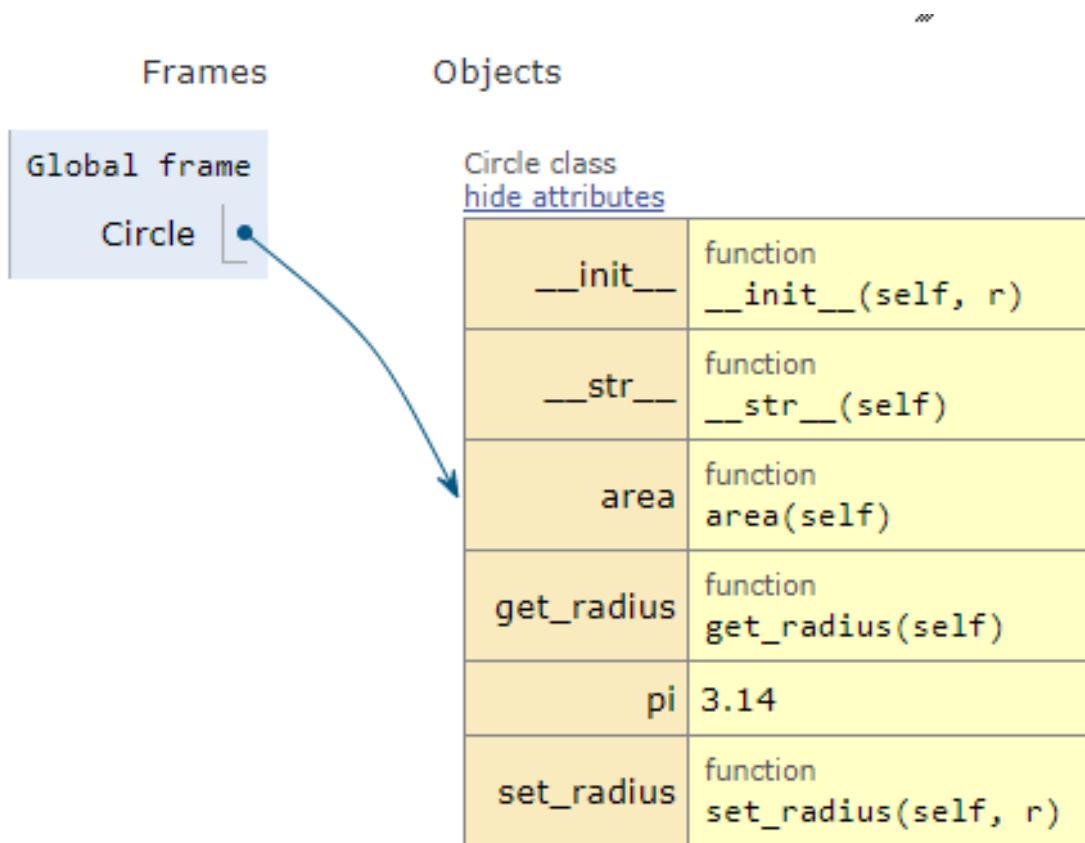
```
In [82]: dir(circle)
Out[82]:
['__doc__',
 '__init__',
 '__module__',
 '__str__',
 'area',
 'get_radius',
 'pi',
 'radius',
 'set_radius']

In [83]:
```

The output of the `dir(circle)` command is shown in blue, while the input and the prompt are in purple.

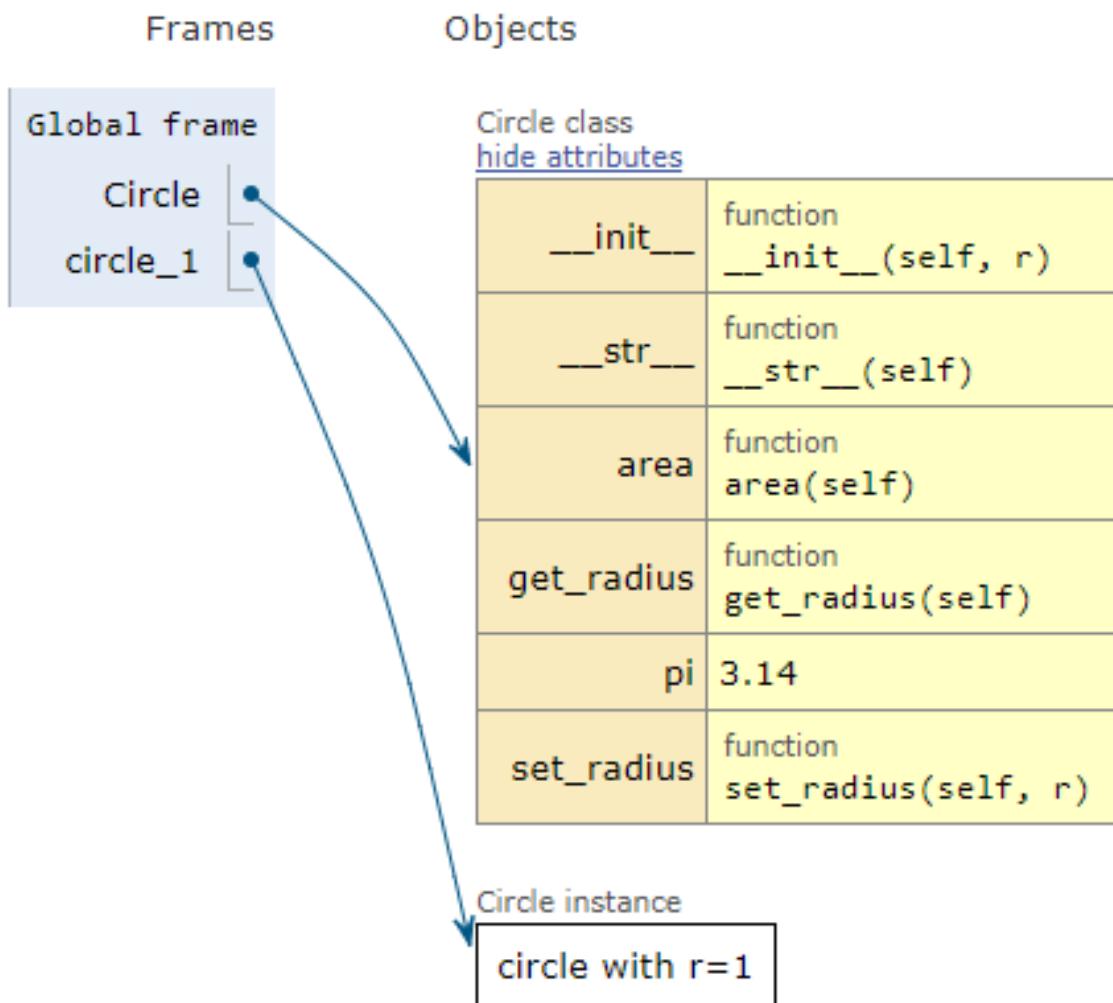
# *Circle()* Class

>>> import circle



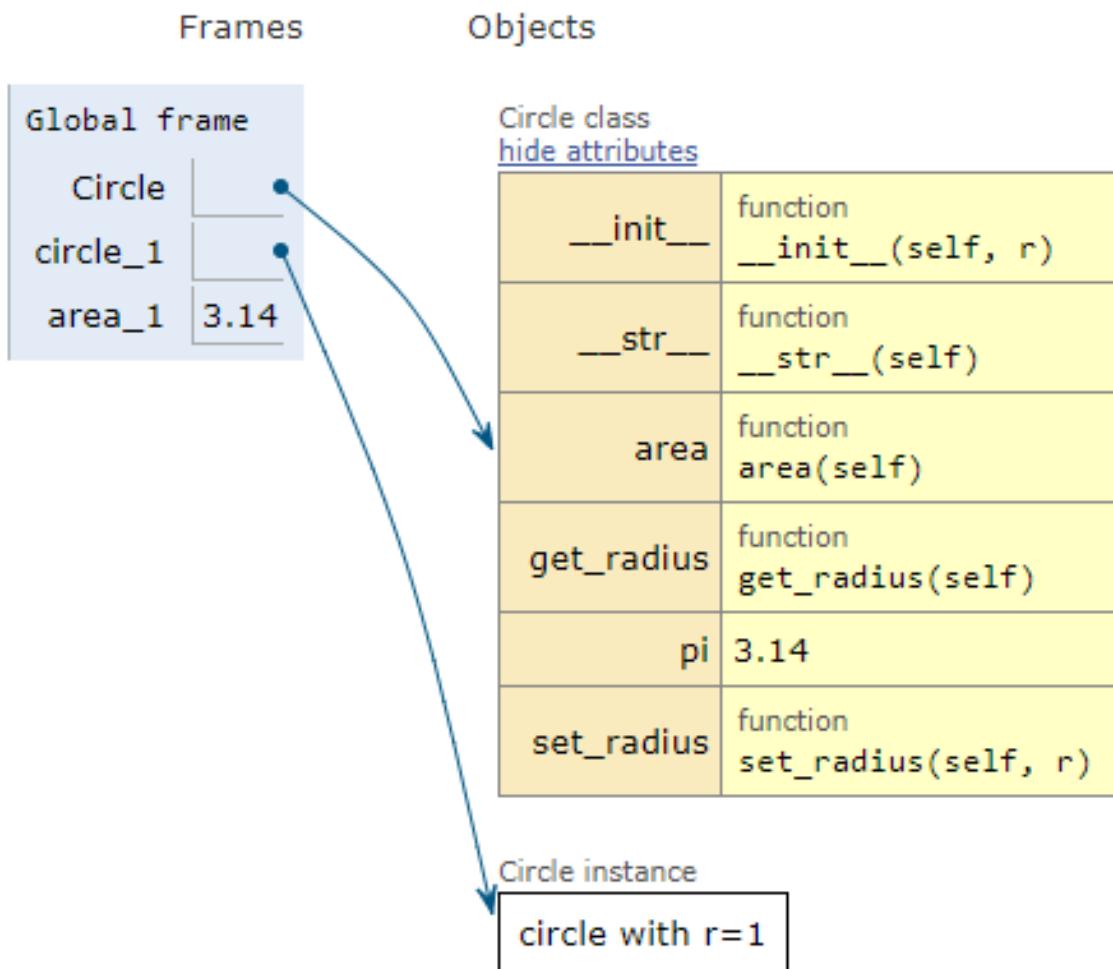
# *Circle()* Class (cont'd)

```
>>> circle_1 = circle.Circle()
```



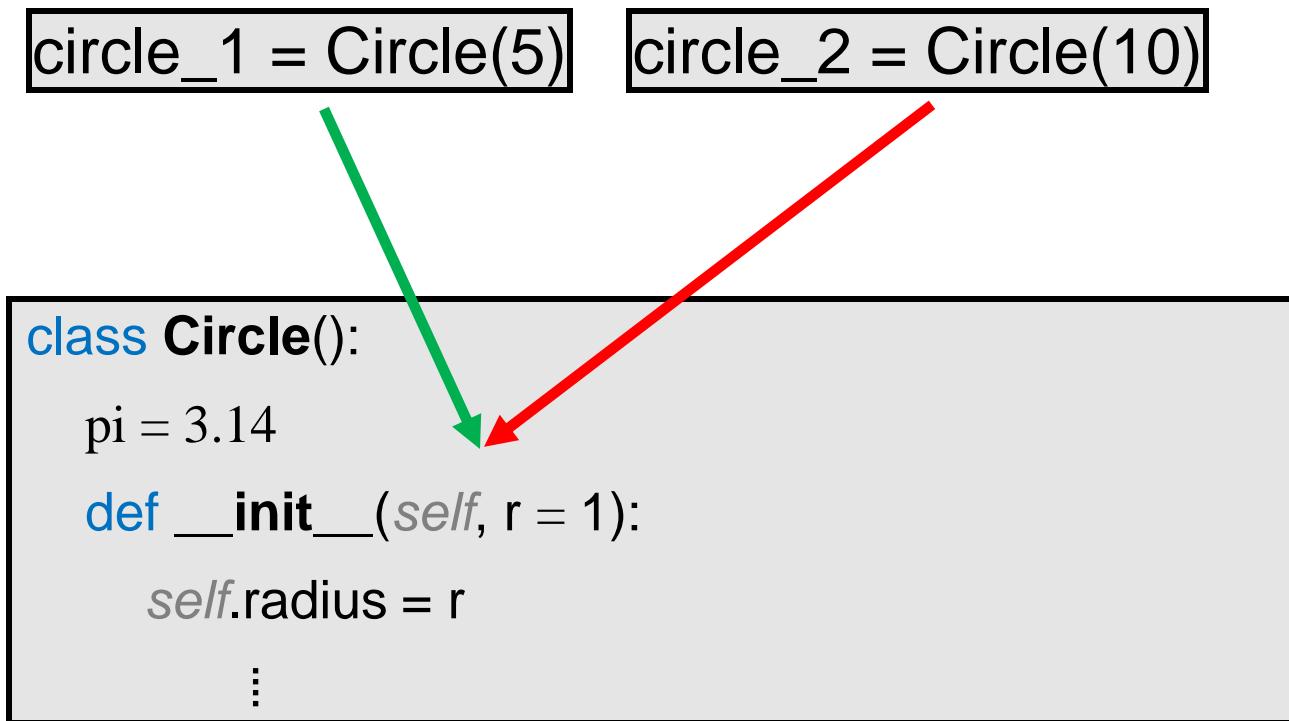
# *Circle()* Class (cont'd)

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



# *self* Parameter

- when an object is initiated, it is passed to the *self* parameter



- allows the object to keep its own data
- similar to *this* in Java and C++

# Constructors and Destructors

- executed when an instance is created
- first parameter is always *self*

```
def __init__(self, r = 1):    # constructor  
    self.radius = r
```

- destructor is executed upon deletion

```
def __del__(self):          # (optional) destructor
```

# Static vs. Non-Static Variables

- non-static variables exist in each class instance (multiple copies)
- static variables are shared (single copy) and are defined before class methods

```
class Circle():
    self.pi = 3.14           # NOT STATIC !
```

```
class Circle():
    pi = 3.14               # STATIC !
```

# Object State Representation:

## `__str__()` and `__repr__()`

- `__repr__()` is a built-in function to compute “official” representation of an object
- `__str__()` gives “human” representation of an object

```
>>> repr(circle)
```

```
>>> '<__main__.Circle instance at  
0x00000000DB7F548>'
```

```
>>> str(circle)
```

```
>>> 'circle r=1'
```

- `__str__()` uses `__repr__()` as a fall-back

# *Circle()* Class Attributes

The screenshot shows an IPython console window with the title bar "IPython console" and a tab labeled "IP: Console 1/A". The console area displays the following Python code and its output:

```
In [90]: Circle.__bases__
Out[90]: ()

In [91]: Circle.__dict__
Out[91]:
{'__doc__': None,
 '__init__': <function __main__.__init__>,
 '__module__': '__main__',
 '__str__': <function __main__.__str__>,
 'area': <function __main__.area>,
 'get_radius': <function __main__.get_radius>,
 'pi': 3.14,
 'set_radius': <function __main__.set_radius>}

In [92]: Circle.__doc__

In [93]: Circle.__module__
Out[93]: '__main__'

In [94]: Circle.__name__
Out[94]: 'Circle'
```

The console window has a scroll bar on the right side. Below the main window is a smaller "Console" window.

# Private Variables

- no mechanism
- textual replacement via name mangling
- \_var replaced by \_classname\_var  
interface for different types

```
>> [name for name in dir(Circle) if  
'Circle' in name]  
['_Circle__pi']
```

- prevents accidental modification

# Private Variables

- make variable private with \_\_
- \_\_radius and \_\_pi are now private
- access private variables with methods

```
class Circle():  
    __pi = 3.14  
    def __init__(self, r = 1):  
        self.__radius = r  
    def __str__(self):  
        s_radius = str(self.__radius)  
        return "circle r=" + s_radius  
    def set_radius(self, r):  
        self.__radius = r  
    def get_radius(self):  
        return self.__radius  
    def area(self):  
        return self.__pi * self.__radius**2
```

# Public vs. Private Variables

- can change variables (pi and radius)
- this is undesirable

The screenshot shows the Spyder IDE interface with the following components:

- Editor:** Displays the code for `circle.py`. The code defines a `Circle` class with methods for initializing radius, getting radius, setting radius, and calculating area. It also creates a `circle` object and prints its area.
- IPython console:** Shows the output of running the code. It includes the command to run the file, the resulting `circle` object, and the calculated area.
- Console:** Shows a note about using the ipython kernel and a warning message.

```
2 class Circle():
3     pi = 3.14      # static
4
5     def __init__(self, r=1):
6         self.radius = r
7
8     def __str__(self):
9         s_radius = str(self.radius)
10        return "circle with r=" + s_radius
11
12    def set_radius(self, r):
13        self.radius = r
14
15    def get_radius(self):
16        return self.radius
17
18    def area(self):
19        return self.pi * self.radius**2
20
21 circle = Circle()
22 circle.pi = 15
23 circle.radius = 10
24 area = circle.area()
25 print(circle)
26 print("area: ", area)
```

```
runfile('D:/bu/python/python_cs521/python_code_examples/circle.py',
wdir='D:/bu/python/python_cs521/python_code_examples')
circle with r=1

In [70]:
runfile('D:/bu/python/python_cs521/python_code_examples/circle.py',
wdir='D:/bu/python/python_cs521/python_code_examples')
circle with r=10
('area: ', 1500)

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

To exit, you will have to expl
```

# *Circle()*: Private Variables

- cannot change variables (pi and radius)
- add class methods to set/get variables

The screenshot shows the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Tools, View, and Help. Below the menu is a toolbar with various icons for file operations like Open, Save, and Run. The main area has tabs for 'circle\_area.py' and 'circle\_protected.py'. The code editor displays the following Python code:

```
1 class Circle():
2     __pi = 3.14      # static and private
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 circle = Circle()
21 circle.pi = 15
22 circle.radius = 10
23 area = circle.area()
24 print(circle)
25 print("area: ", area)
26
```

To the right of the code editor is the IPython console window, which contains three entries:

- In [72]:  
runfile('D:/bu/python/python\_cs521/python\_code\_examples/circle\_protected.py',  
wdir='D:/bu/python/python\_cs521/python\_code\_examples')  
circle with r=1  
('area: ', 3.14)
- In [72]:  
runfile('D:/bu/python/python\_cs521/python\_code\_examples/circle\_protected.py',  
wdir='D:/bu/python/python\_cs521/python\_code\_examples')  
circle with r=1  
('area: ', 3.14)
- In [73]:

Below the IPython console is a smaller 'Console' window with the message:

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

At the bottom of the interface, status bars show 'Permissions: RW', 'End-of-lines: CRLF', 'Encoding ASCII', 'Line: 2', 'Colu 41', 'Memory: 52 %', and a progress bar.

# Accessing and Setting Class Variables

- data field encapsulation
- expose instance variables by methods
- accessors: return values w/o changes
- mutators: set and/or change values

```
def set_radius(self, r):      # mutator  
    self.radius = r           # (setter)
```

```
def get_radius(self):        # accessor  
    return self.radius       # (getter)
```

# Pre-defined Class Attributes

attribute	description	type
<code>__bases__</code>	classes from which this class inherits	tuple of classes
<code>__dict__</code>	class name space	dictionary
<code>__doc__</code>	class documentation	None/string
<code>__module__</code>	name of class module	string
<code>__name__</code>	class name	string

- instance attributes are created by methods using *self* parameter

# Object Documentation

```
def area(self):  
    return self.pi * self.radius**2
```

```
>>> print circle.area.__doc  
>>> None  
>>> # now add docstring as first line
```

```
def area(self):  
    """ computing area """  
    return self.pi * self.radius**2
```

```
>>> print circle.area.__doc  
>>> computing area
```

# Review Problems

# Interview Problem

- explain the characteristics of Python's objects

# Interview Problem

- how is a Python class created?

# Interview Problem

- what is `__init__.py` file?

# Interview Problem

- what is encapsulation?

# Interview Problem

- what is `repr()` function and what does it return?

# Interview Problem

- what does the `self` keyword do?

# Interview Problem

- how is a class instantiated?

# Interview Problem

- in a class definition, what does the `__init__()` function do?

# Interview Problem

- what is the output of the following:

```
class C(object)
    def __init__(self)
        self.x = 1
```

```
>>> c = C()
>>> print c.x
>>> print c.x
>>> print c.x
```

# Interview Problem

- what is a class?
- how do you create class in Python?

# Interview Problem

- how instance variables are different from class variables?

# Interview Problem

- what are accessors, mutators,  
@property?

# Interview Problem

- define a protected member in class

# Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT'. To the right of these are the words 'Workbench' and icons for help ('?') and settings ('gear'). Below this is the title 'Exercise 51078 —'. Underneath the title are two tabs: 'WORK AREA' (which is selected) and 'SOLUTIONS'. A green button labeled 'Content Support' is located above the main content area. The main content area contains the following text:

Suppose there is a class AirConditioner. The class supports the following behaviors: turning the air conditioner on and off. The following methods are provided for these behaviors: turn\_on and turn\_off. Both methods accept no arguments and return no value.

There is a reference variable my\_ac to an object of this class, which has already been created. Invoke a method to turn the air conditioner on.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a help icon with a gear symbol on the right. Below the navigation bar, the title 'Exercise 51079 —' is displayed. Underneath the title, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located below the tabs. The main content area contains the following text:

Suppose there is a class AirConditioner. The class supports the following behaviors: turning the air conditioner on and off. The following **methods** are provided for these behaviors: `turn_on` and `turn_off`. Both methods accept **no arguments** and **return no value**.

There is a reference variable `my_ac` to an **object** of this class, which has already been created. Invoke a method to turn the air conditioner **off**.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a software interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a question mark icon and settings gear icon on the right. Below the navigation bar, the title 'Exercise 51080 —' is displayed. Underneath the title, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located below the tabs. The main content area contains the following text:

Suppose there is a class `AirConditioner`. The class supports the following behaviors: turning the air conditioner on, off, and setting the desired temperature. The following **methods** are provided for these behaviors: `turn_on` and `turn_off`, which accept **no arguments** and **return no value**, and `set_temp`, which accepts an **int argument** and **returns no value**.

There is a reference `variable my_ac` to an object of this class, which has already been created. Invoke a method telling the object to set the air conditioner to 72 degrees.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a 'Workbench' interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, and a help/gear icon on the right. The title 'Workbench' is centered above a section labeled 'Exercise 51081 —'. Below this, there are two tabs: 'WORK AREA' (highlighted in orange) and 'SOLUTIONS'. A green button labeled 'Content Support' is located in the top right corner of the main content area. The main content area contains the following text:

Suppose there is a class `AirConditioner`. The class supports the following **behaviors**: turning the air conditioner on, off, setting the desired temperature, and telling the previously set temperature. The following **methods** are provided for these behaviors: `turn_on` and `turn_off`, which accept **no arguments** and **return no value**, `set_temp`, which accepts an int argument and **returns no value**, and `get_temp`, which accepts **no value** and returns an int.

There is a reference variable `my_ac` to an **object** of this class, which has already been created. There is also a variable called `current_temp`, which has already been initialized. Invoke a **method** using the reference variable which returns the previously set temperature of the air conditioner, and store the returned value in `current_temp`.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a software interface titled "Workbench". At the top left are "PREV" and "NEXT" buttons. In the top right are a help icon (?) and a settings gear icon. The title "Exercise 51082" is displayed above a toolbar with "WORK AREA" and "SOLUTIONS" tabs. A green button labeled "Content Support" is visible. The main content area contains the following text:

Suppose there is a class `AirConditioner`. The class supports the following behaviors: turning the air conditioner on, off, and checking if the air conditioner is on or off. The following **methods** are provided for these behaviors: `turn_on` and `turn_off`, which accept **no arguments** and **return no value**, and `is_on`, which accepts **no argument** and returns a bool indicating whether the air conditioner is on or off.

There is a reference variable `my_ac` to an **object** of this class, which has already been created. There is also a variable `status`, which has already been initialized which refers to a bool. Invoke a method of this object using the reference variable, asking the object to tell whether the air conditioner is on or off, and store the result in `status`.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a help icon ('?') and settings icon ('⚙️') on the right. Below these is the title 'Exercise 51083 —'. Underneath the title are two tabs: 'WORK AREA' (highlighted in blue) and 'SOLUTIONS'. A green button labeled 'Content Support' is located below the tabs. The main content area contains the following text:

Suppose there is a class AirConditioner. The class supports the following behaviors: turning the air conditioner on and off. The following **methods** are provided for these behaviors: `turn_on` and `turn_off`. Both methods accept **no arguments** and **return no value**.

There is a reference `variable office_a_c` of type AirConditioner. Create a **new object** of type AirConditioner using the `office_a_c` reference `variable`. After that, turn the air conditioner **on** using the reference to the new object.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a software interface for a programming exercise. At the top, there are navigation buttons labeled 'PREV' and 'NEXT', a title 'Workbench', and a help icon. Below the title, the exercise number 'Exercise 51084' is displayed. A tab bar at the bottom has 'WORK AREA' and 'SOLUTIONS' tabs, with 'WORK AREA' currently selected. A 'Content Support' button is located above the main content area. The main content area contains the following text:

Suppose there is a class AirConditioner. The class supports the following behaviors: turning the air conditioner on, off, and setting the desired temperature. The following **methods** are provided for these behaviors: `turn_on` and `turn_off`, which accept **no arguments** and **return no value**, and `set_temp`, which accepts an **int argument** and **returns no value**.

There is a reference variable `office_a_c` of type `AirConditioner`. Create a **new object** of type `AirConditioner` using the `office_a_c` reference variable. After that, turn the air conditioner **on** using the reference to the new object, and set the desired temperature to 69 degrees.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a user interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a question mark icon and a gear icon on the right. Below the navigation bar, the title 'Exercise 51225 —' is displayed. Underneath the title, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located above the main content area. The main content area contains the following text and list:

Write the definition of a class Counter containing:

- An instance variable counter of type int, initialized to 0.
- A method called increment that adds one to the instance variable counter. It does not accept parameters or return a value.
- A method called get\_value that doesn't accept any parameters. It returns the value of the instance variable counter.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a software interface for a programming exercise. At the top, there are navigation buttons: 'PREV' and 'NEXT' on the left, 'Workbench' in the center, and a question mark icon and a gear icon on the right. Below the navigation bar, the title 'Exercise 51226 —' is displayed. Underneath the title, there are two tabs: 'WORK AREA' and 'SOLUTIONS'. A green button labeled 'Content Support' is located above the main content area. The main content area contains the following text and list:

Write the definition of a class Counter containing:

- An instance variable named counter of type int.
- A constructor that takes one int argument and assigns its value to counter
- A method named increment that adds one to counter. It does not take parameters or return a value.
- A method named decrement that subtracts one from counter. It also does not take parameters or return a value.
- A method named get\_value that returns the value of the instance variable counter.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a software interface for a programming exercise. At the top, there are navigation buttons labeled 'PREV' and 'NEXT' with arrows, followed by the title 'Workbench'. To the right are help and settings icons. Below the title, it says 'Exercise 51227 —'. There are two tabs: 'WORK AREA' (which is selected) and 'SOLUTIONS'. A 'Content Support' button is located above the main text area. The main text area contains the following instructions:

Write the definition of a class Counter containing:

- An instance variable named counter of type int
- An instance variable named limit of type int.
- A constructor that takes two int arguments and assigns the first one to counter and the second one to limit
- A method named increment. It does not take parameters or return a value; if the instance variable counter is less than limit, increment just adds one to the instance variable counter.
- A method named decrement. It also does not take parameters or return a value; if counter is greater than zero, it just subtracts one from the counter.
- A method named get\_value that returns the value of the instance variable counter.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a software interface for a programming exercise. At the top, there are navigation buttons labeled 'PREV' and 'NEXT' with arrows, followed by the word 'Workbench'. To the right are help and settings icons. Below the title, 'Exercise 51244' is displayed. The interface is divided into two tabs: 'WORK AREA' and 'SOLUTIONS', with 'WORK AREA' currently selected. A 'Content Support' button is located above the main text area. The main text area contains the following instructions:

Write the definition of a class Player containing:

- An instance **variable** name of type String, initialized to the empty String.
- An instance **variable** score of type int, initialized to zero.
- A method called **set\_name** that has one parameter, whose value it assigns to the instance **variable** name.
- A method called **set\_score** that has one parameter, whose value it assigns to the instance **variable** score.
- A method called **get\_name** that has no parameters and that returns the value of the instance **variable** name.
- A method called **get\_score** that has no parameters and that returns the value of the instance **variable** score.

No constructor need be defined.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a 'Workbench' interface for a programming exercise. At the top, there are navigation buttons for 'PREV' and 'NEXT', the title 'Workbench', and a help/gear icon. Below the title, it says 'Exercise 51245 —'. There are two tabs: 'WORK AREA' (which is selected) and 'SOLUTIONS'. In the bottom right corner of the main area, there is a 'Content Support' button. The main content area contains the following text and list:

Write the definition of a class ContestResult containing:

- An instance variable winner of type String, initialized to the empty String.
- An instance variable second\_place of type String, initialized to the empty String.
- An instance variable third\_place of type String, initialized to the empty String.
- A method called set\_winner that has one parameter, whose value it assigns to the instance variable winner.
- A method called set\_second\_place that has one parameter, whose value it assigns to the instance variable second\_place.
- A method called set\_third\_place that has one parameter, whose value it assigns to the instance variable third\_place.
- A method called get\_winner that has no parameters and that returns the value of the instance variable winner.
- A method called get\_second\_place that has no parameters and that returns the value of the instance variable second\_place.
- A method called get\_third\_place that has no parameters and that returns the value of the instance variable third\_place.

No constructor need be defined.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**

# Programming Exercise

The screenshot shows a programming exercise interface. At the top, there are navigation buttons: 'PREV' and 'NEXT'. To the right of these is the title 'Workbench'. Further right are icons for help (?) and settings (gear). Below the title, the text 'Exercise 51247' is displayed. Underneath the title, there are two tabs: 'WORK AREA' (which is selected) and 'SOLUTIONS'. In the main content area, there is a 'Content Support' button. The task description reads: 'Write the definition of a class WeatherForecast that provides the following behavior (methods):'. Below this, a bulleted list specifies the required methods: 'set\_skies', 'set\_high', 'set\_low', 'get\_skies', 'get\_high', and 'get\_low'. A note at the bottom states: 'No constructor need be defined. Be sure to define instance variables as needed by your "get"/"set" methods.'

Exercise 51247

Workbench

PREV NEXT

Content Support

WORK AREA SOLUTIONS

Write the definition of a class WeatherForecast that provides the following behavior (methods):

- A method called set\_skies that has one parameter, a String.
- A method called set\_high that has one parameter, an int.
- A method called set\_low that has one parameter, an int.
- A method called get\_skies that has no parameters and that returns the value that was last used as an argument in set\_skies.
- A method called get\_high that has no parameters and that returns the value that was last used as an argument in set\_high.
- A method called get\_low that has no parameters and that returns the value that was last used as an argument in set\_low.

No constructor need be defined. Be sure to define instance variables as needed by your "get"/"set" methods.

# **Programming Exercise**

## **Worksheet**

**this page is intentionally left blank**