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
```

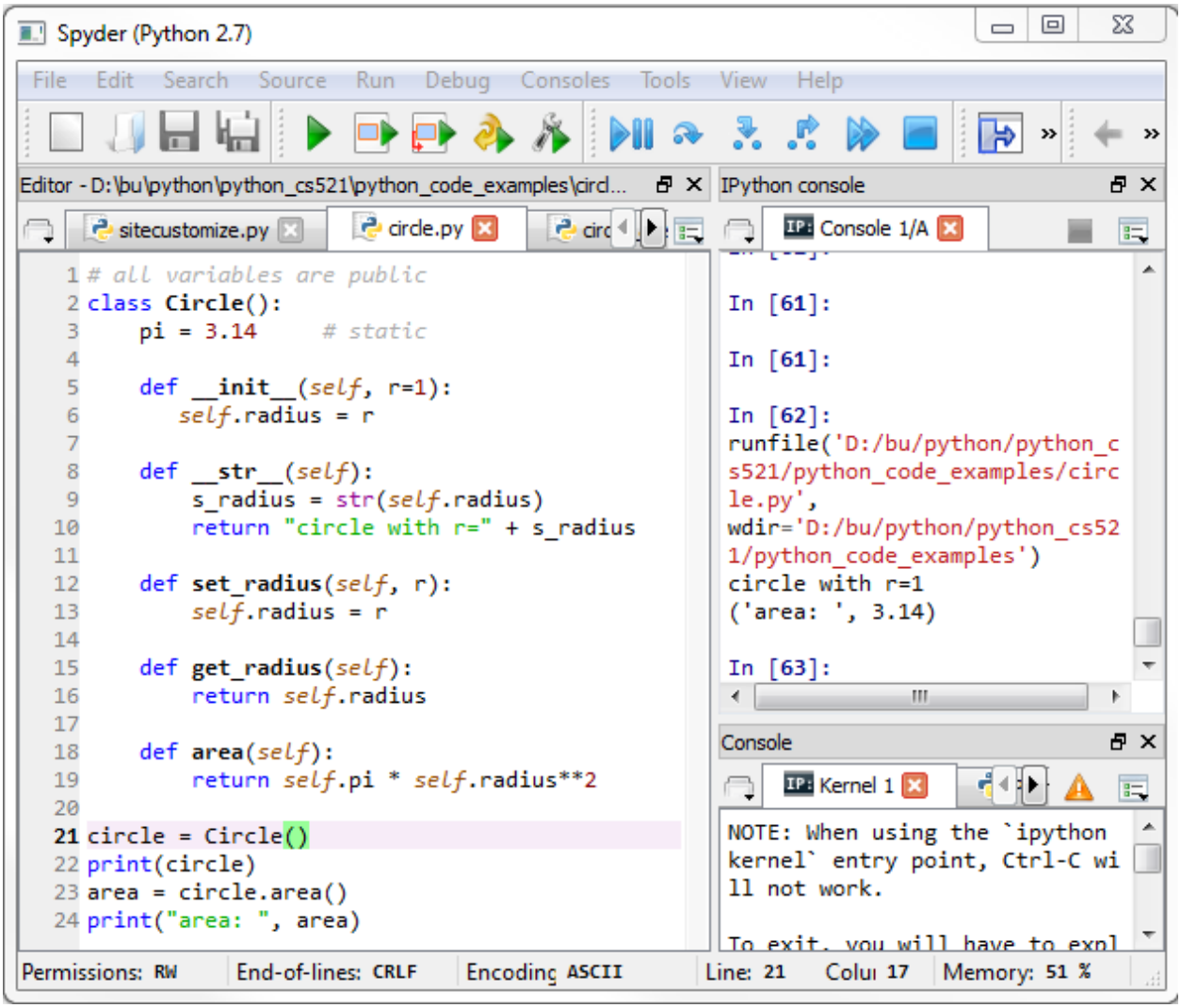
```
    pi = 3.14                   # (static) data fields
```

```
                                #    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()
```

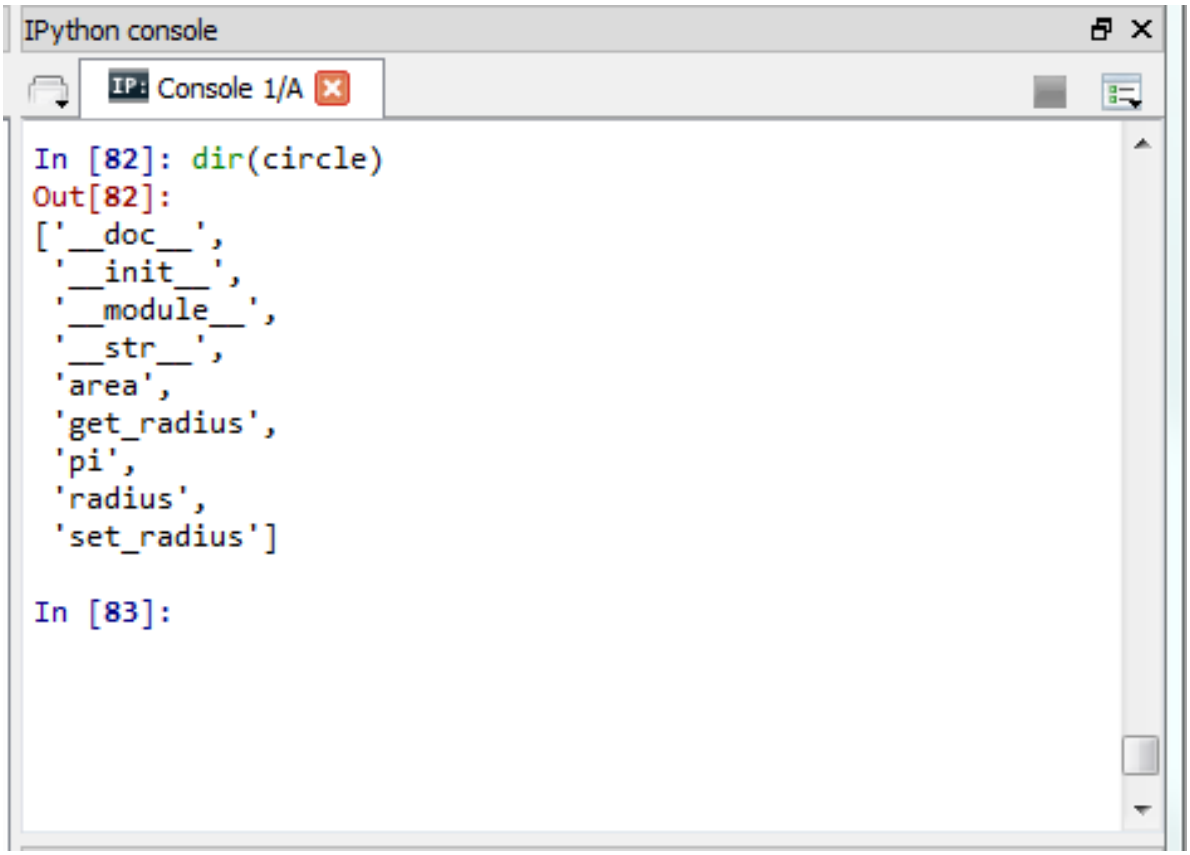
Base *Circle*() Class Run



Class Methods and Variables

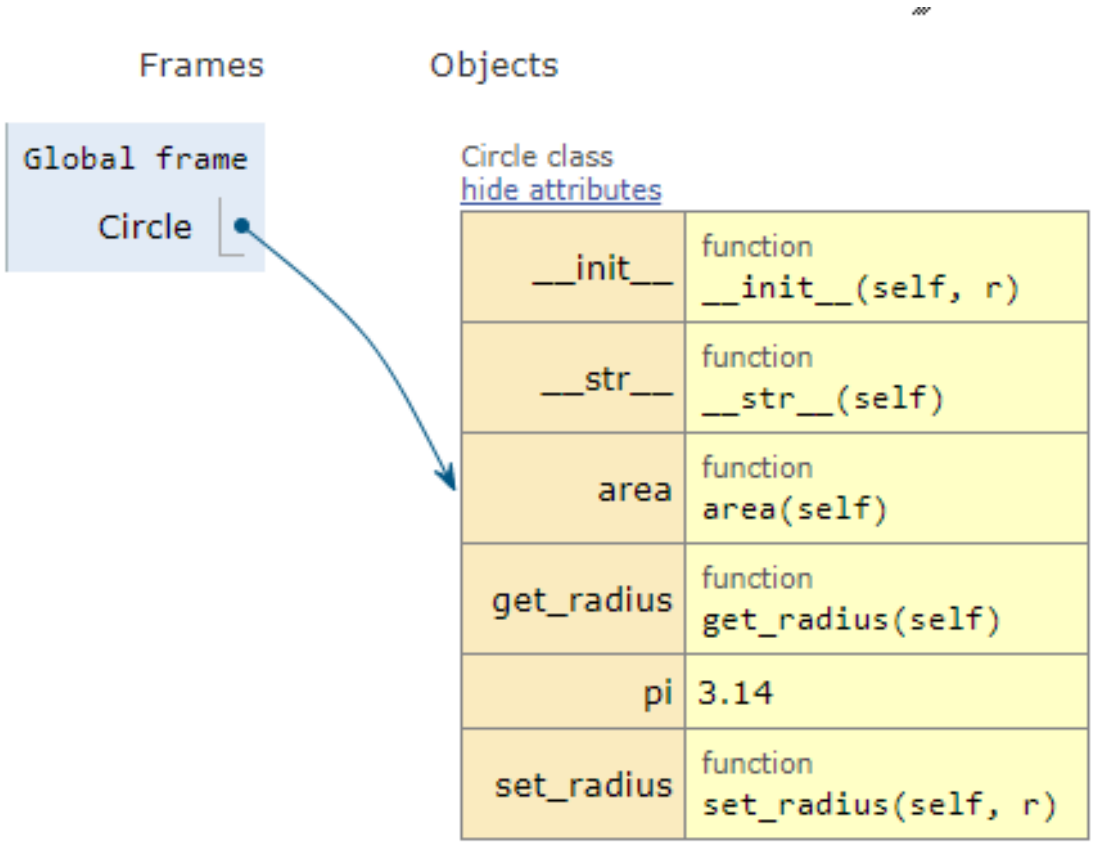
- use dir() command

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



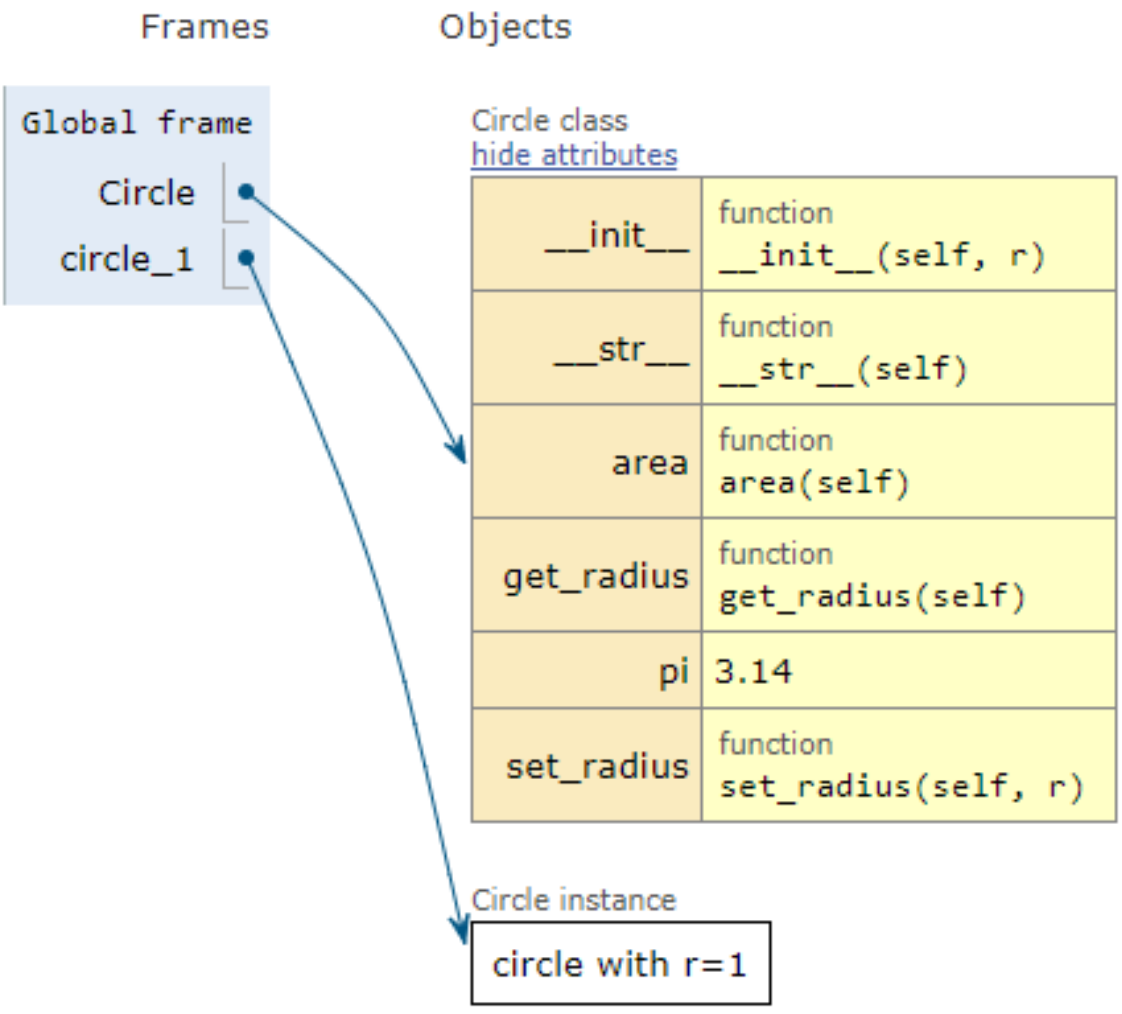
Circle() **Class**

```
>>> import circle
```



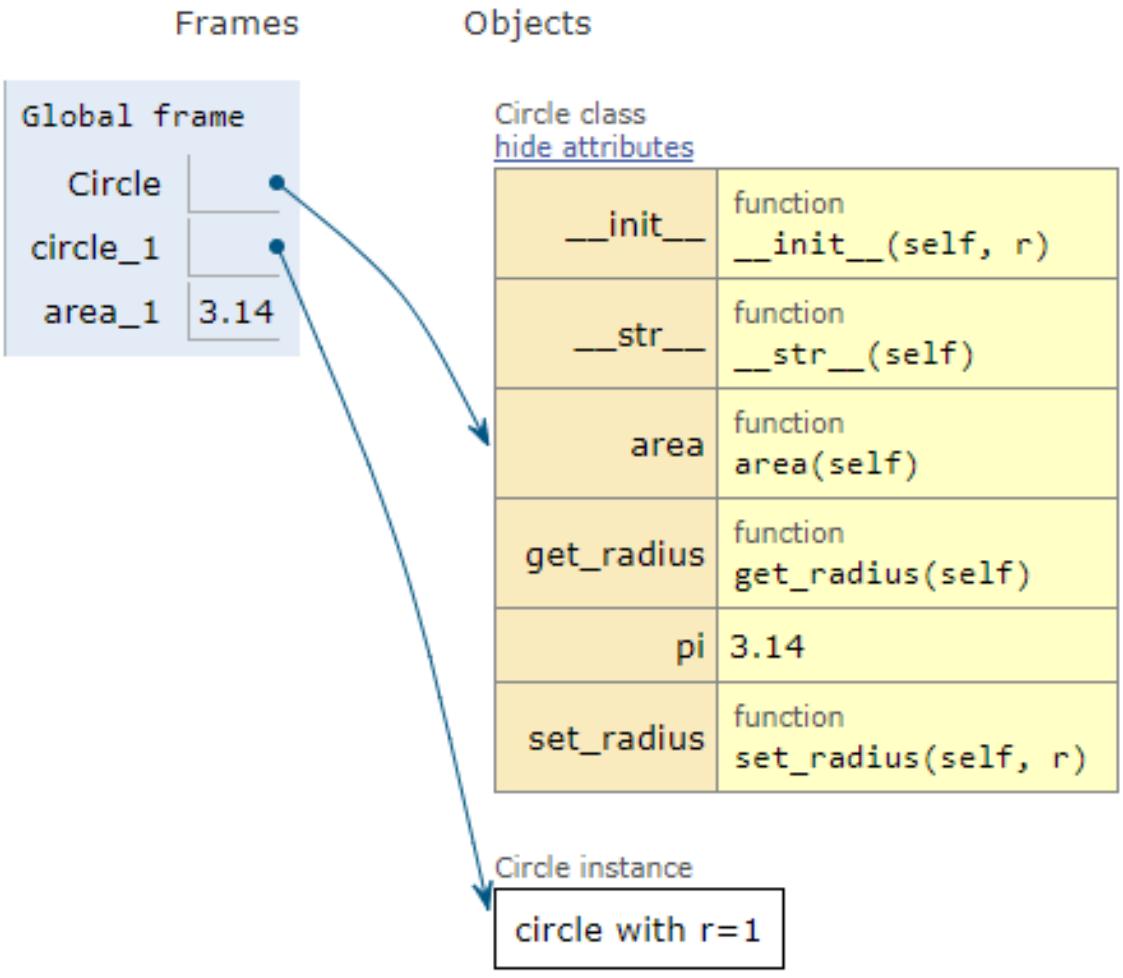
Circle() Class (cont'd)

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



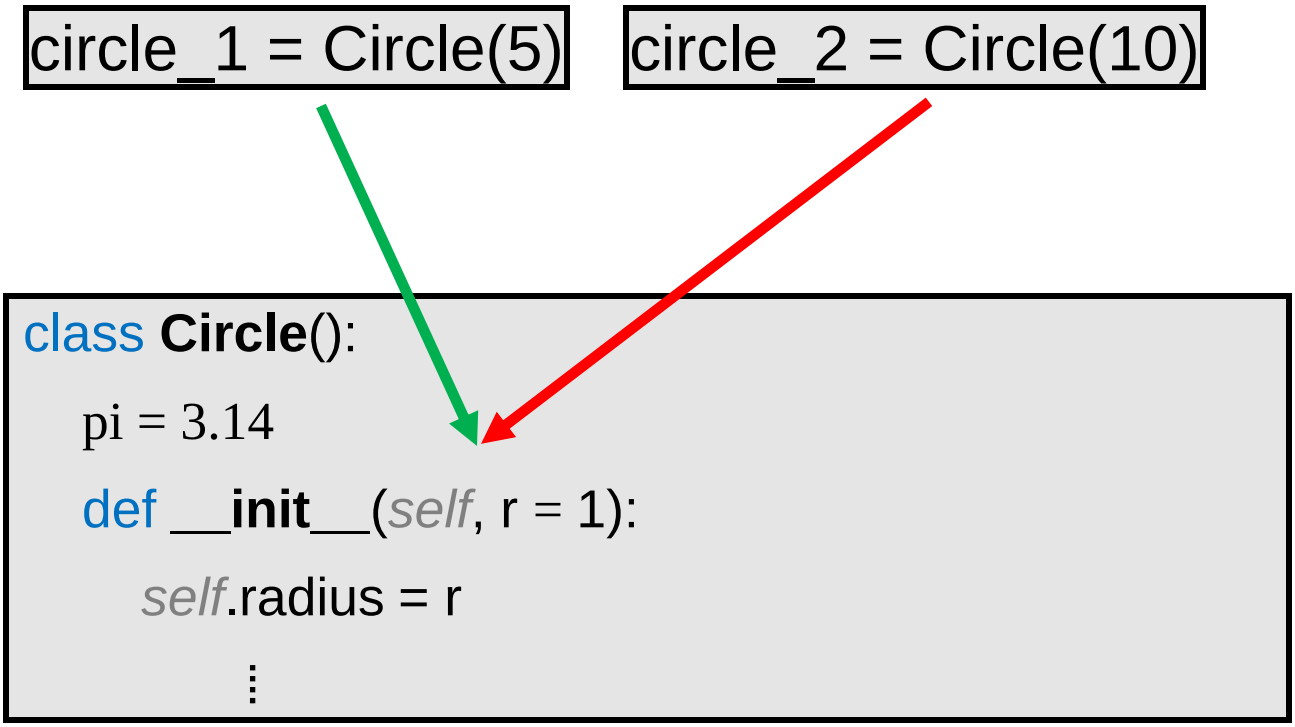
Circle() **Class (cont'd)**

```
>>> 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:

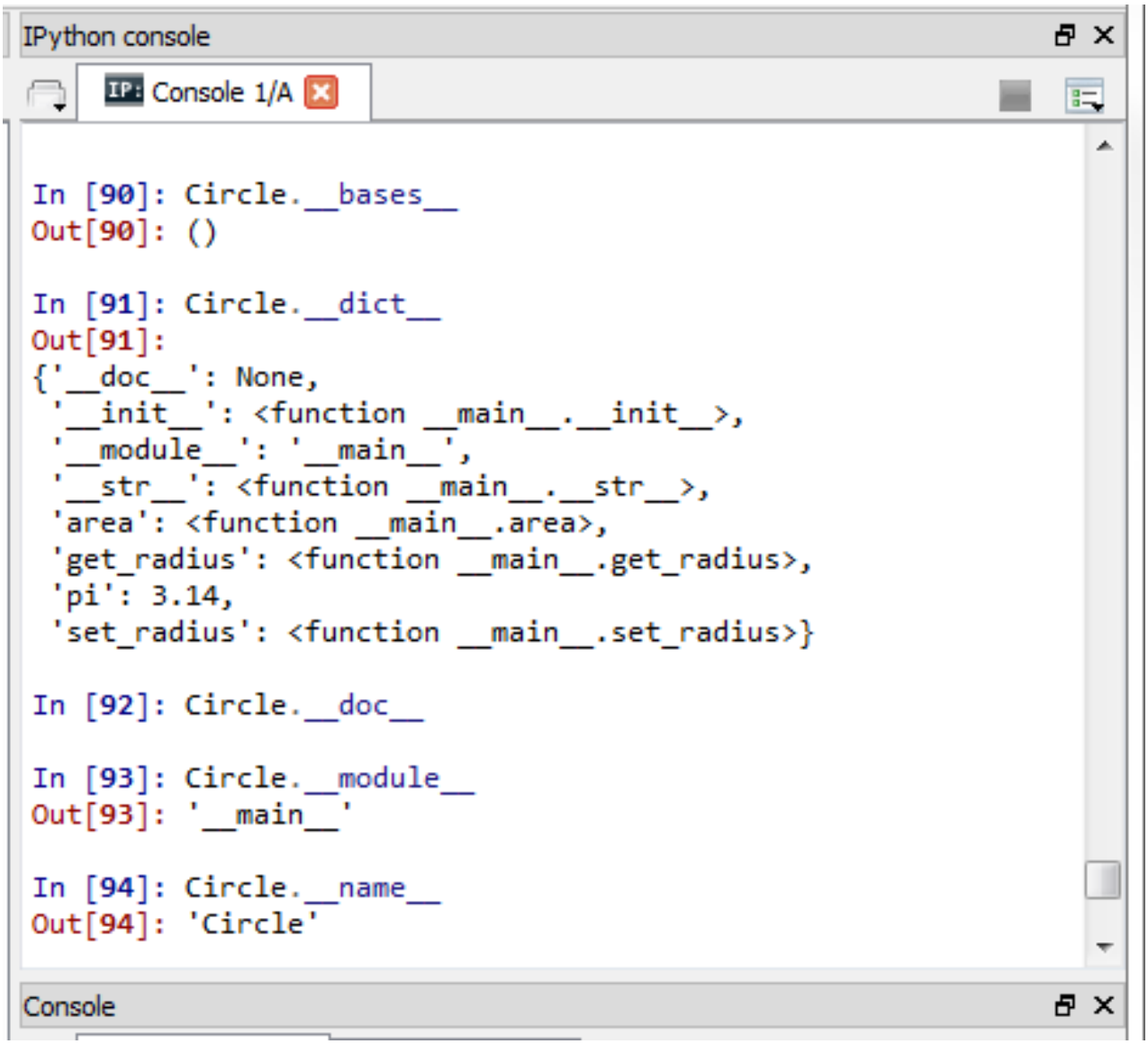
__*str*__() and __*repr*__()

- __*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 0x000000000DB7F548>'
>>> str(circle)
>>> 'circle r=1'
```

- __*str*__() uses __*repr*__() as a fall-back

Circle() Class Attributes



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

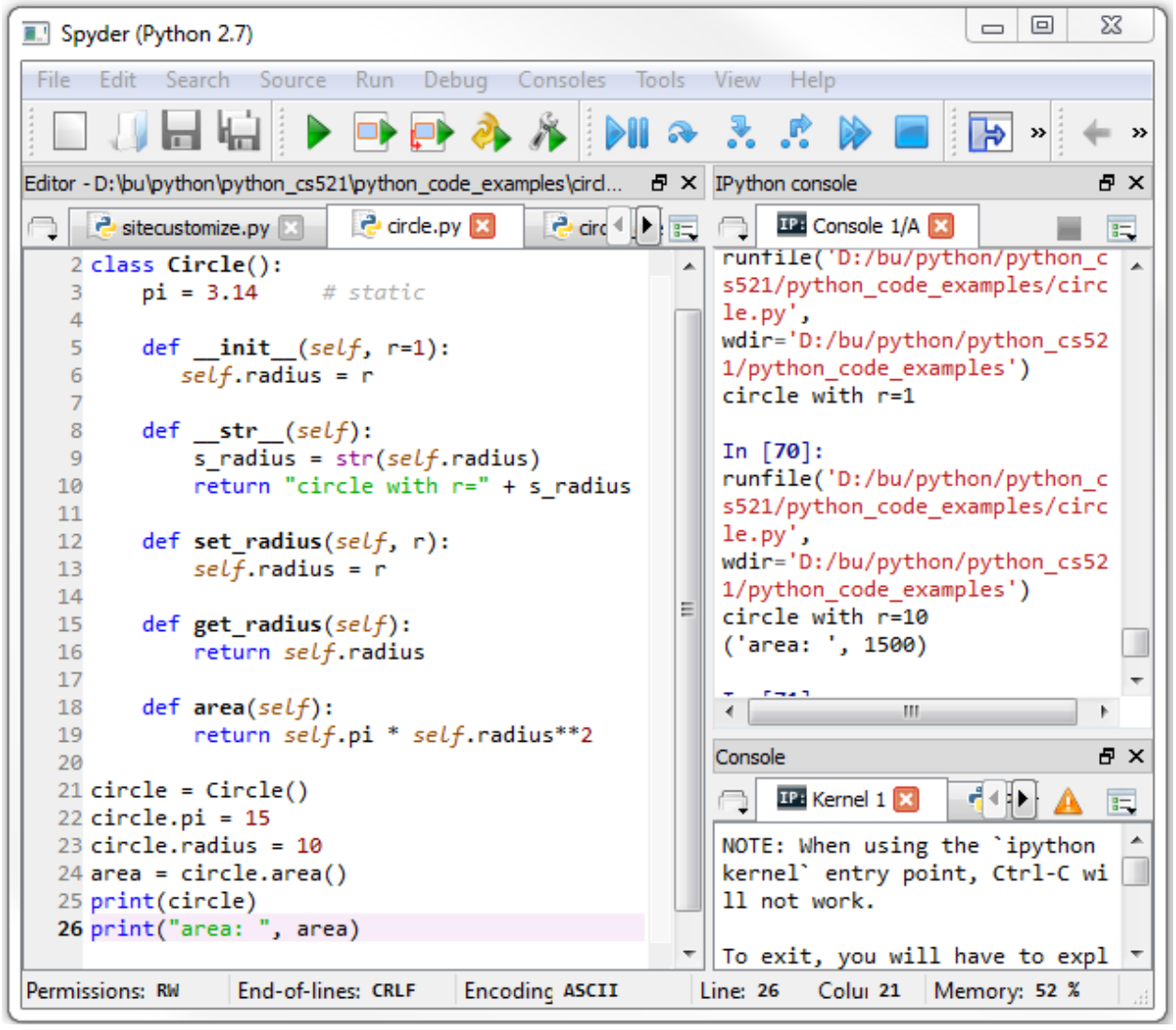
Private Variables

- 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
```

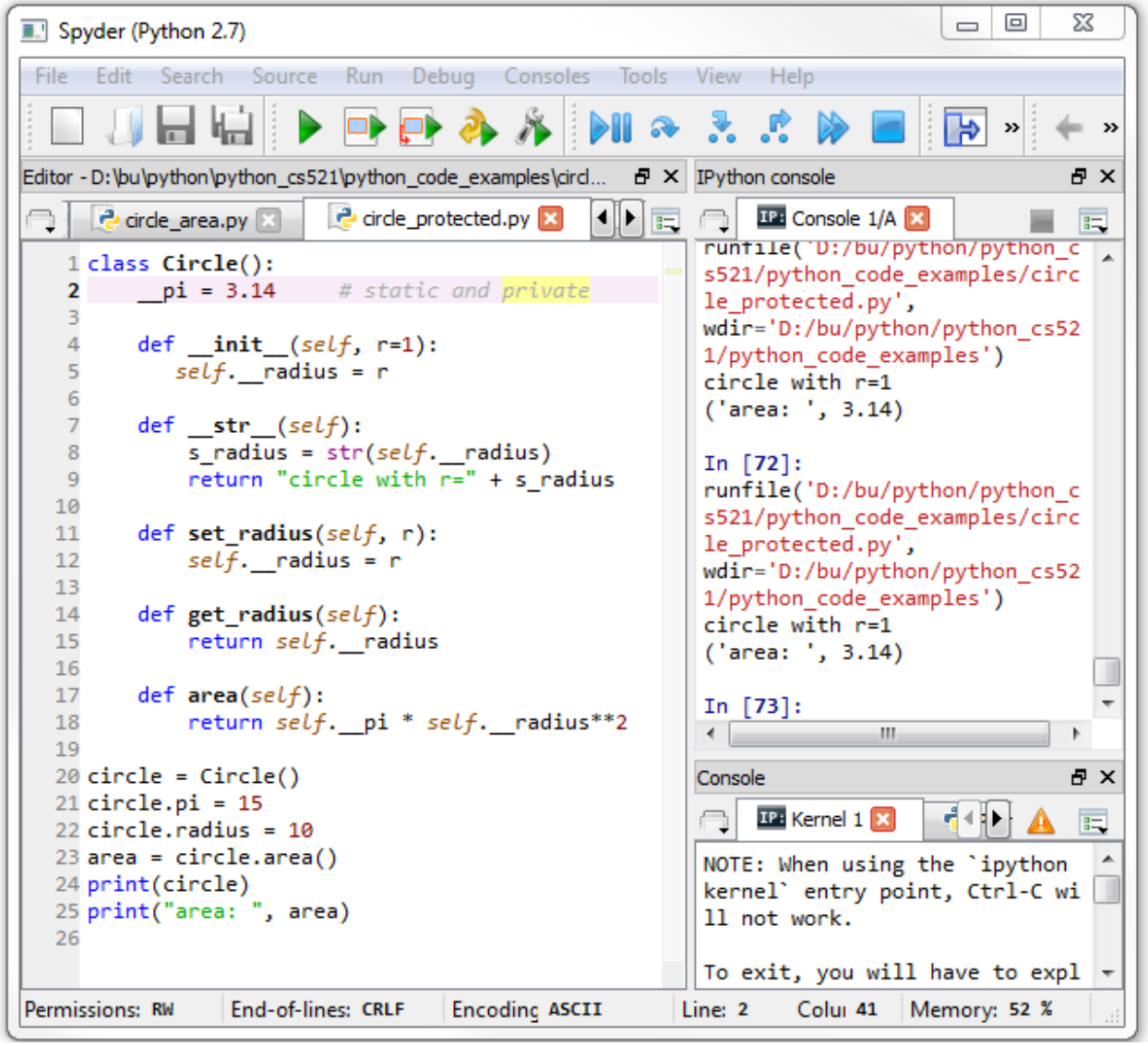
Public vs. Private Variables

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



Circle(): Private Variables

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



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
__bases__	classes from which this class inherits	tuple of classes
__dict__	class name space	dictionary
__doc__	class documentation	None/string
__module__	name of class module	string
__name__	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?

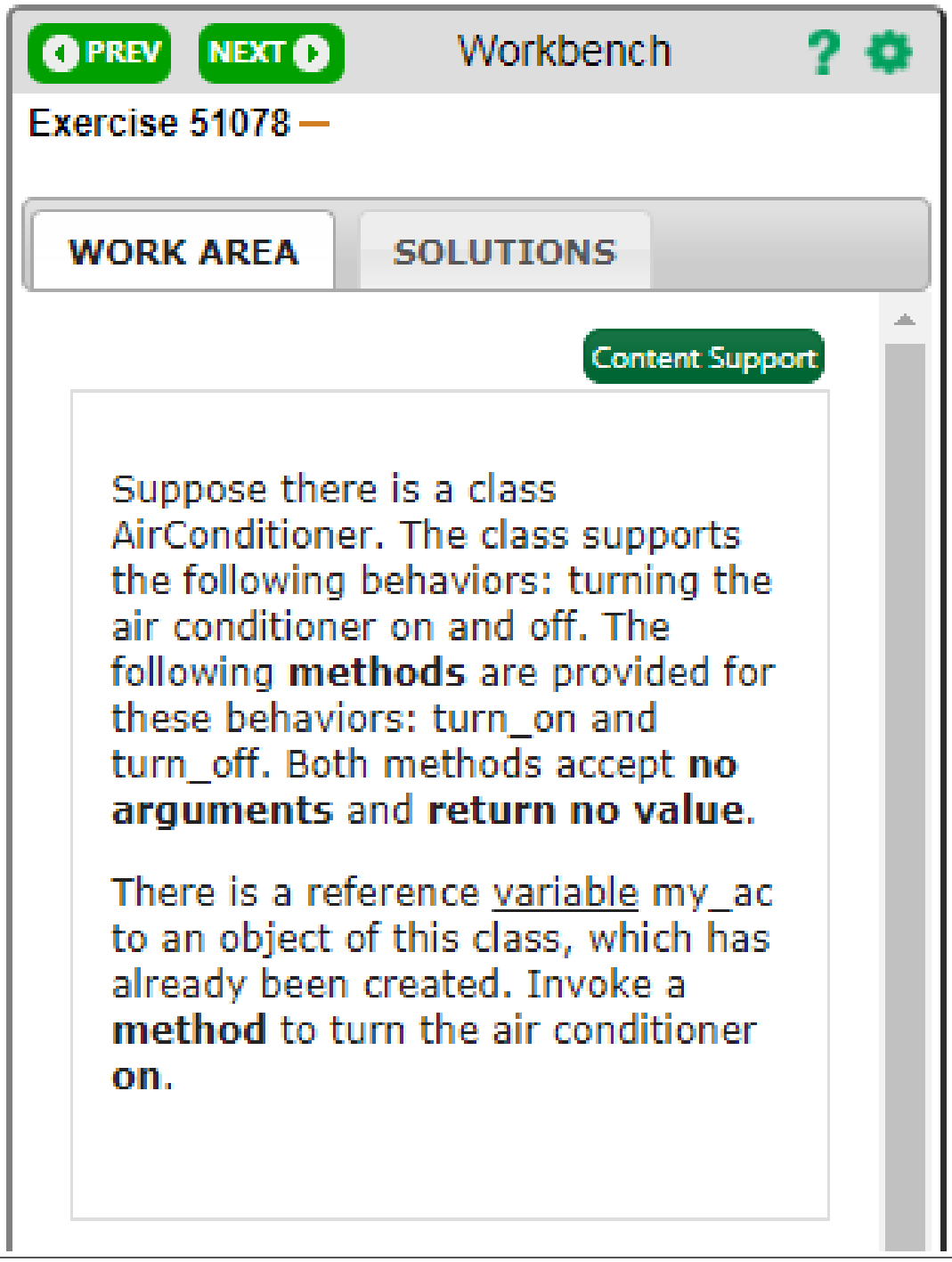
Interview Problem

- what are accessors, mutators, @property?

Interview Problem

- define a protected member in class

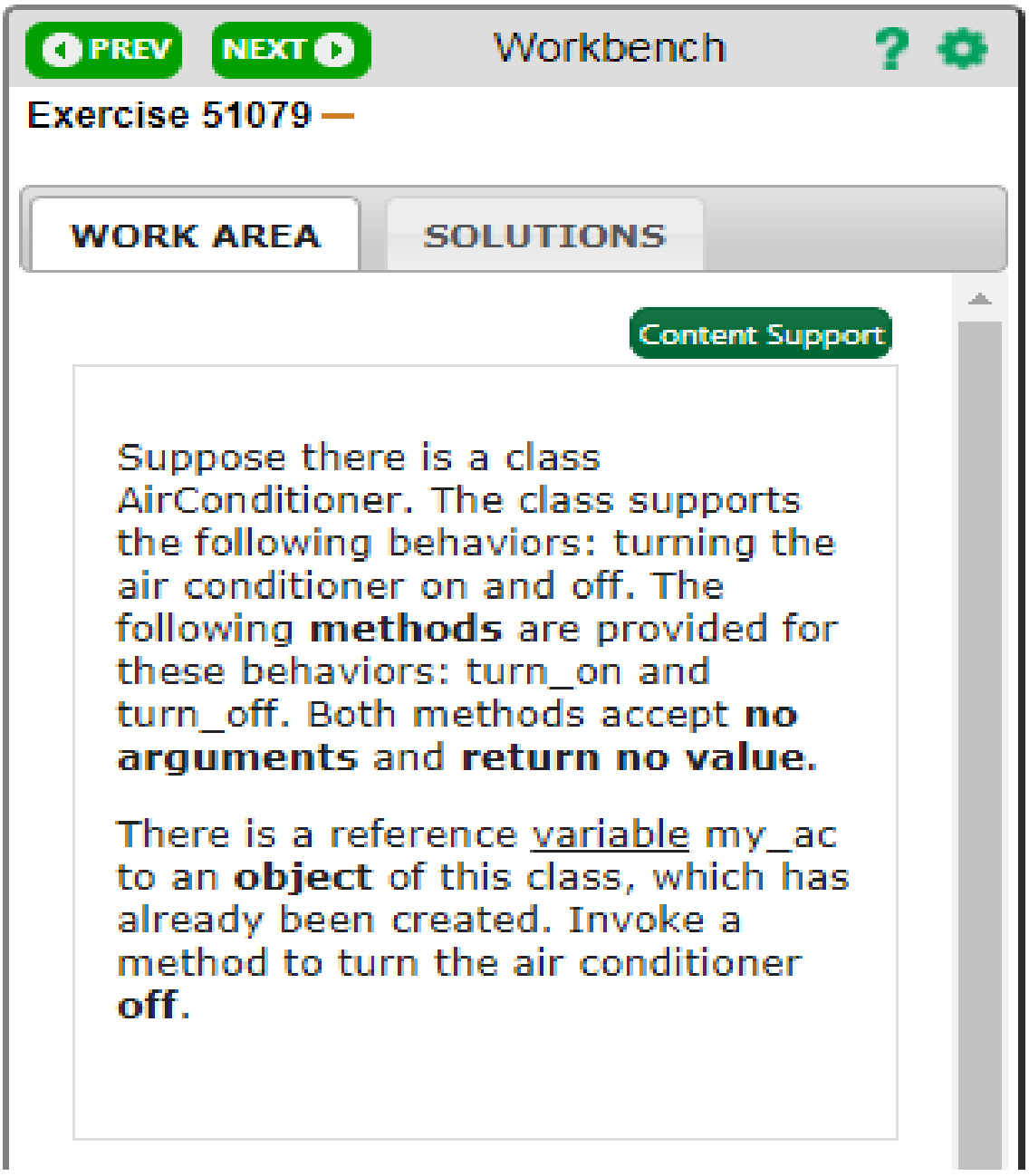
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

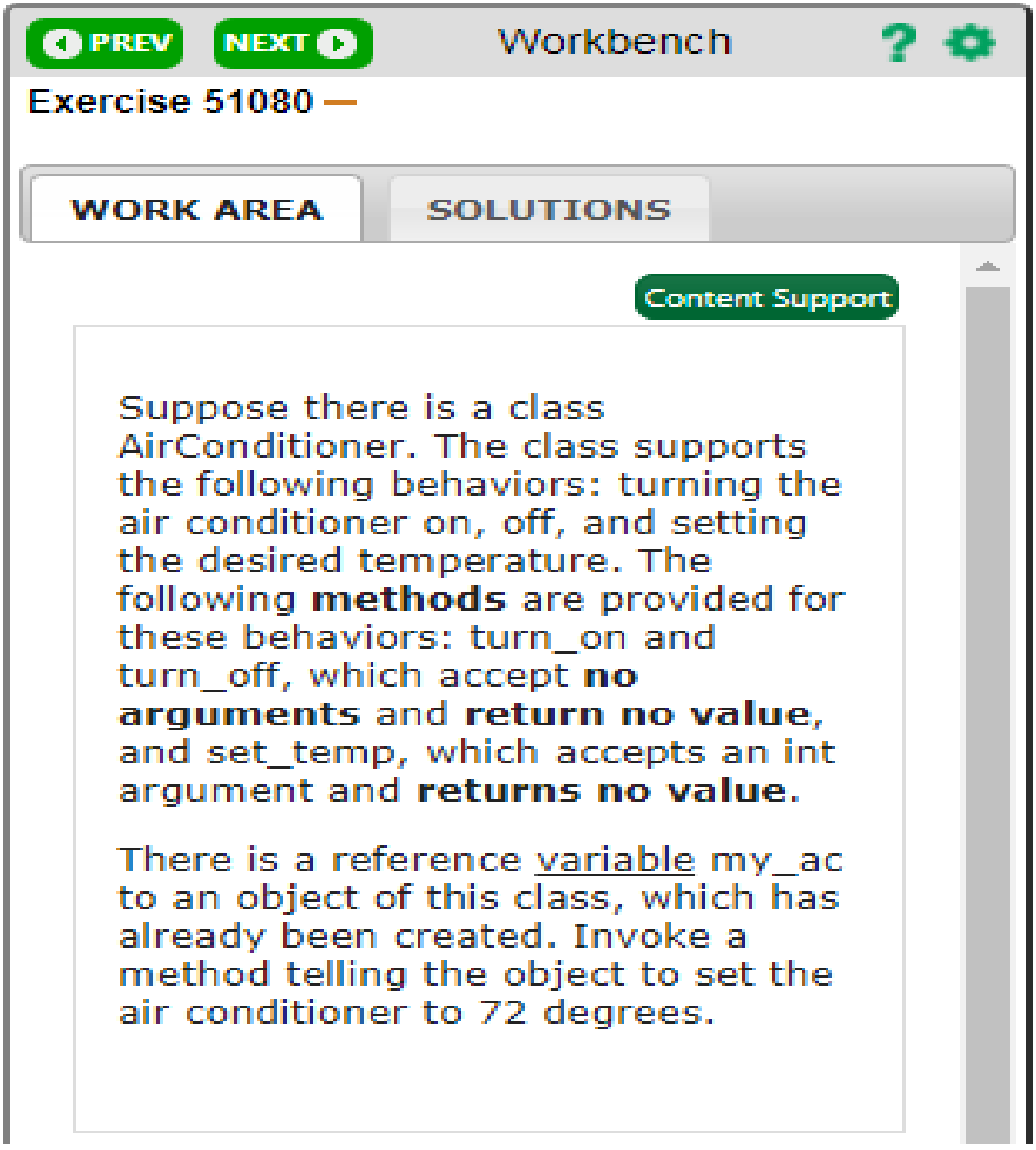
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

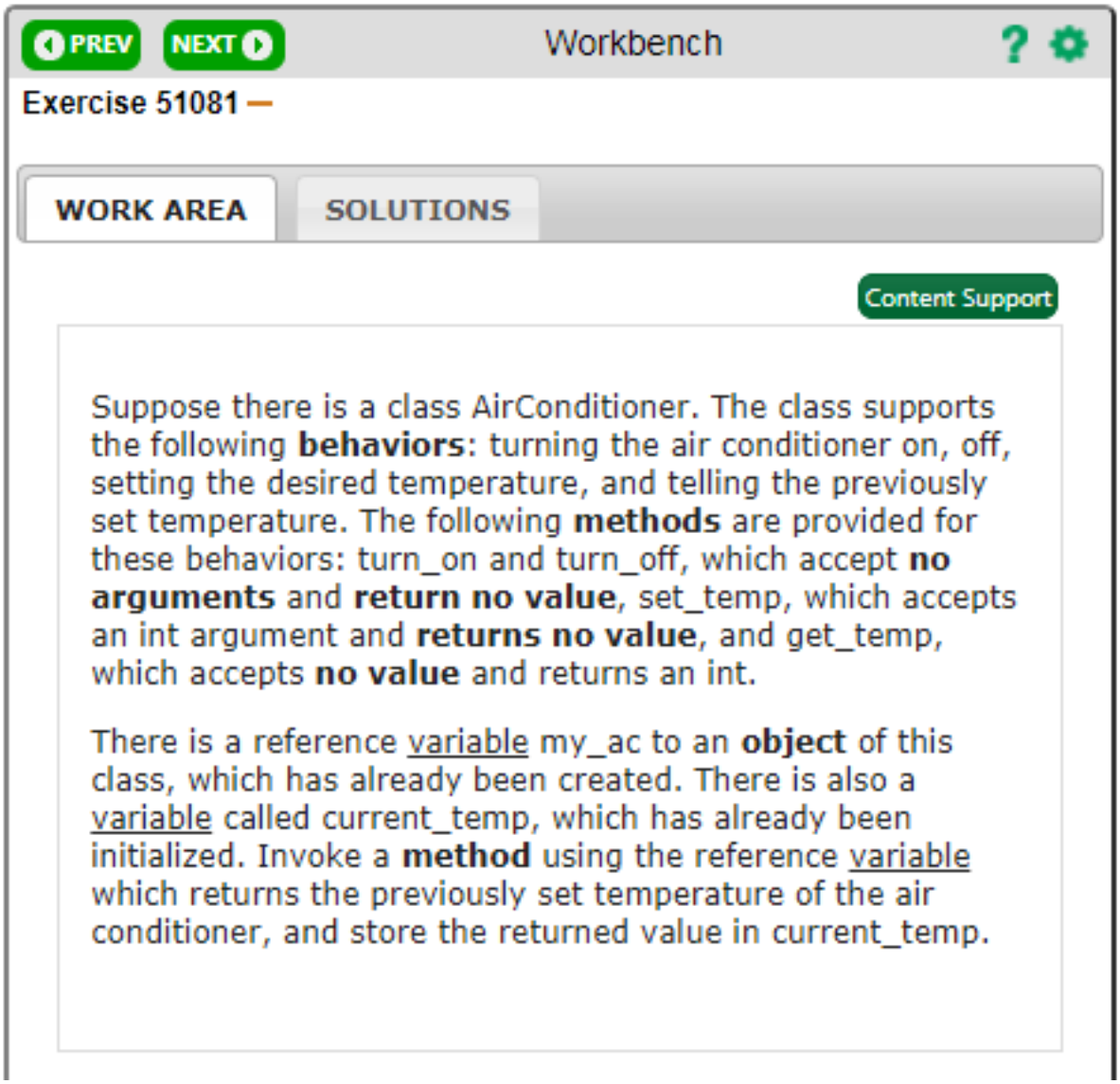
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

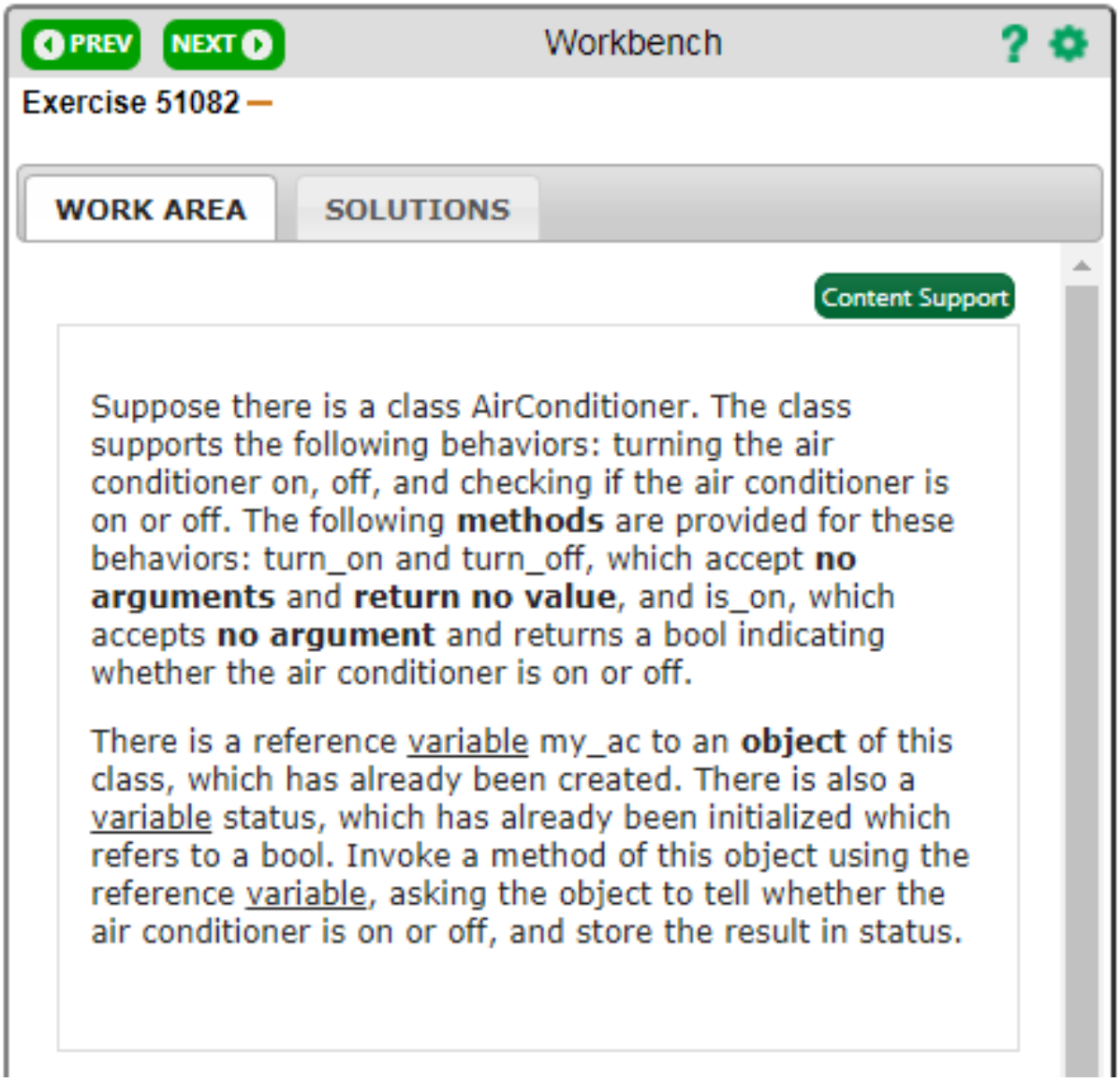
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

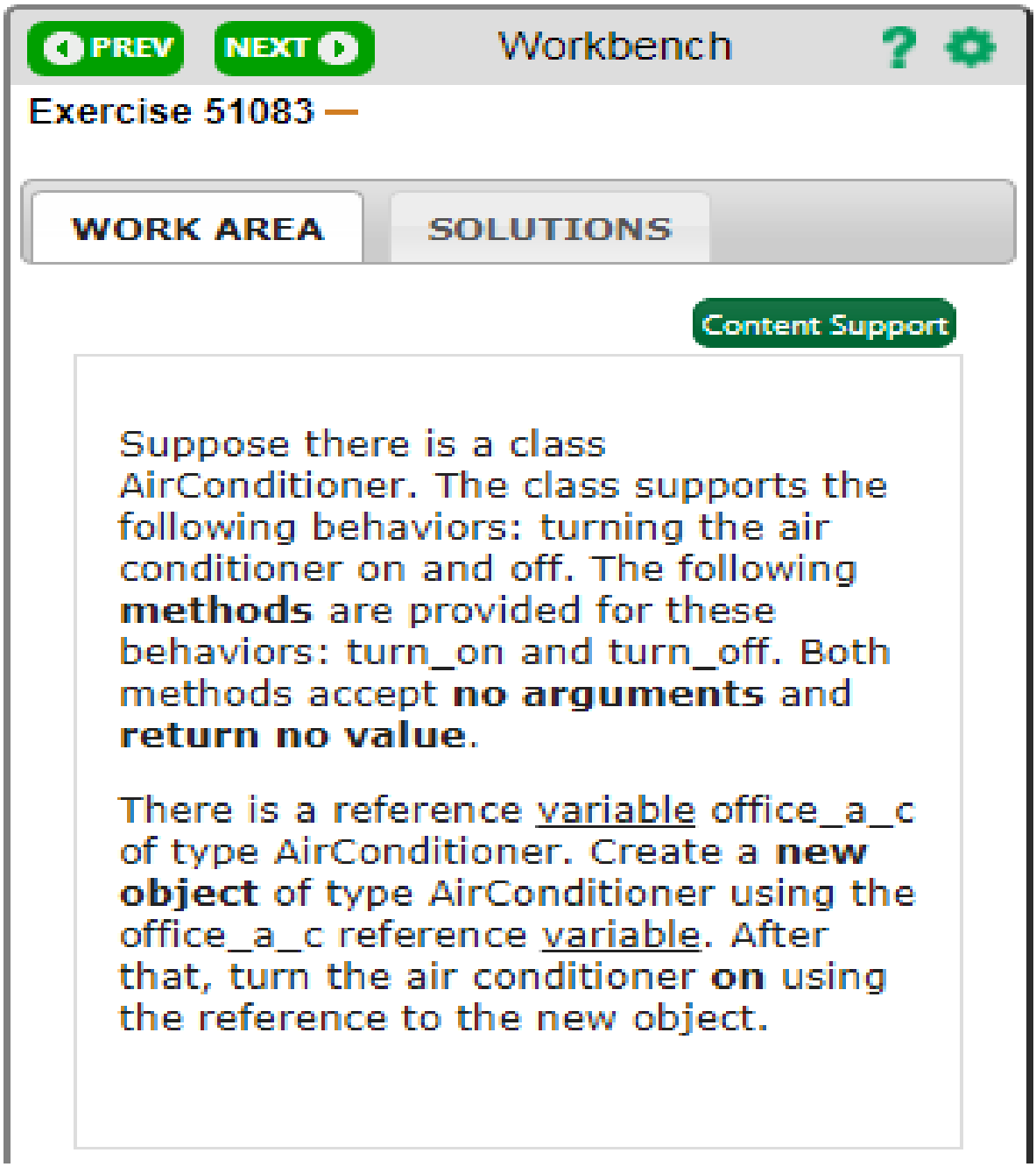
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

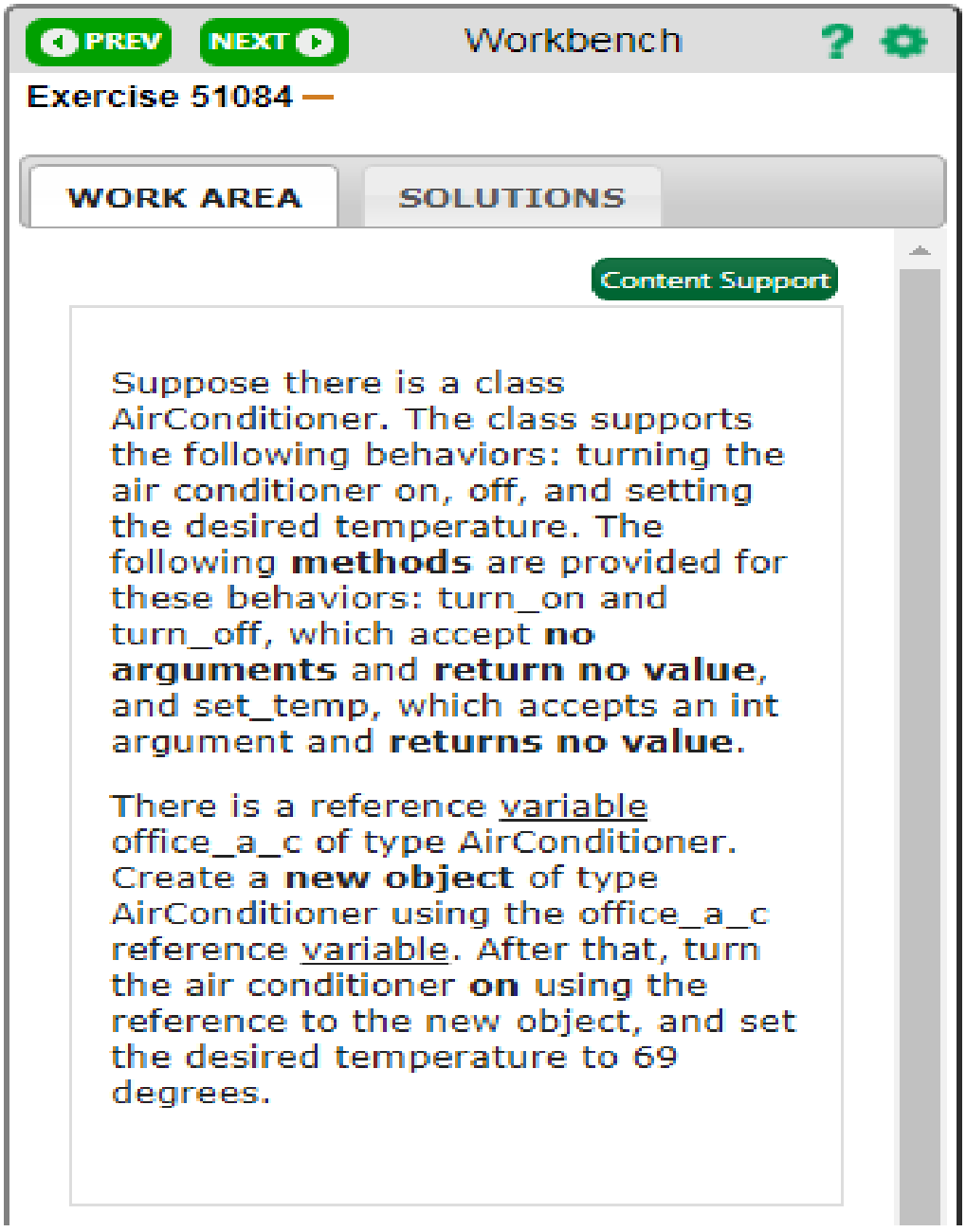
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

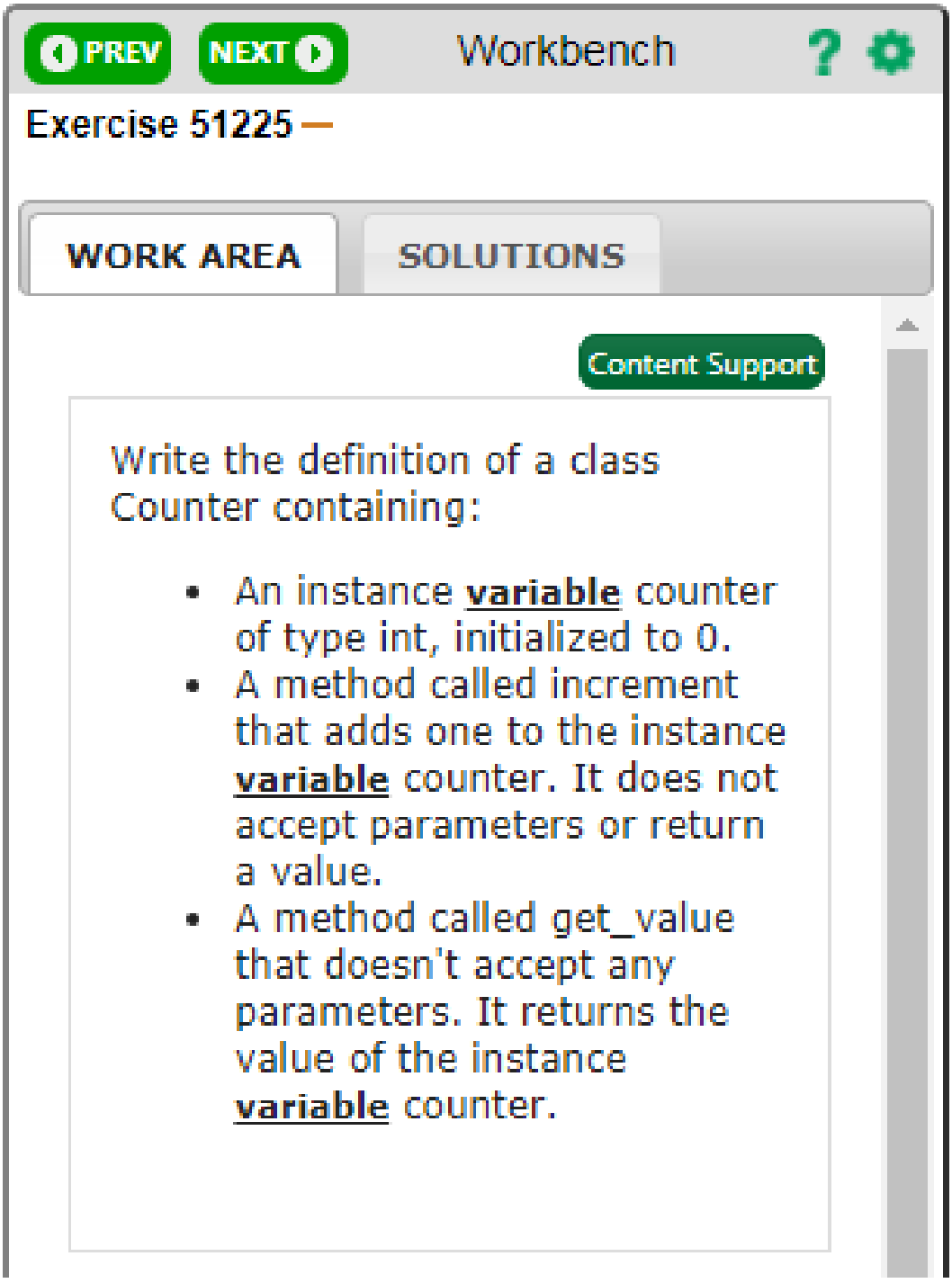
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

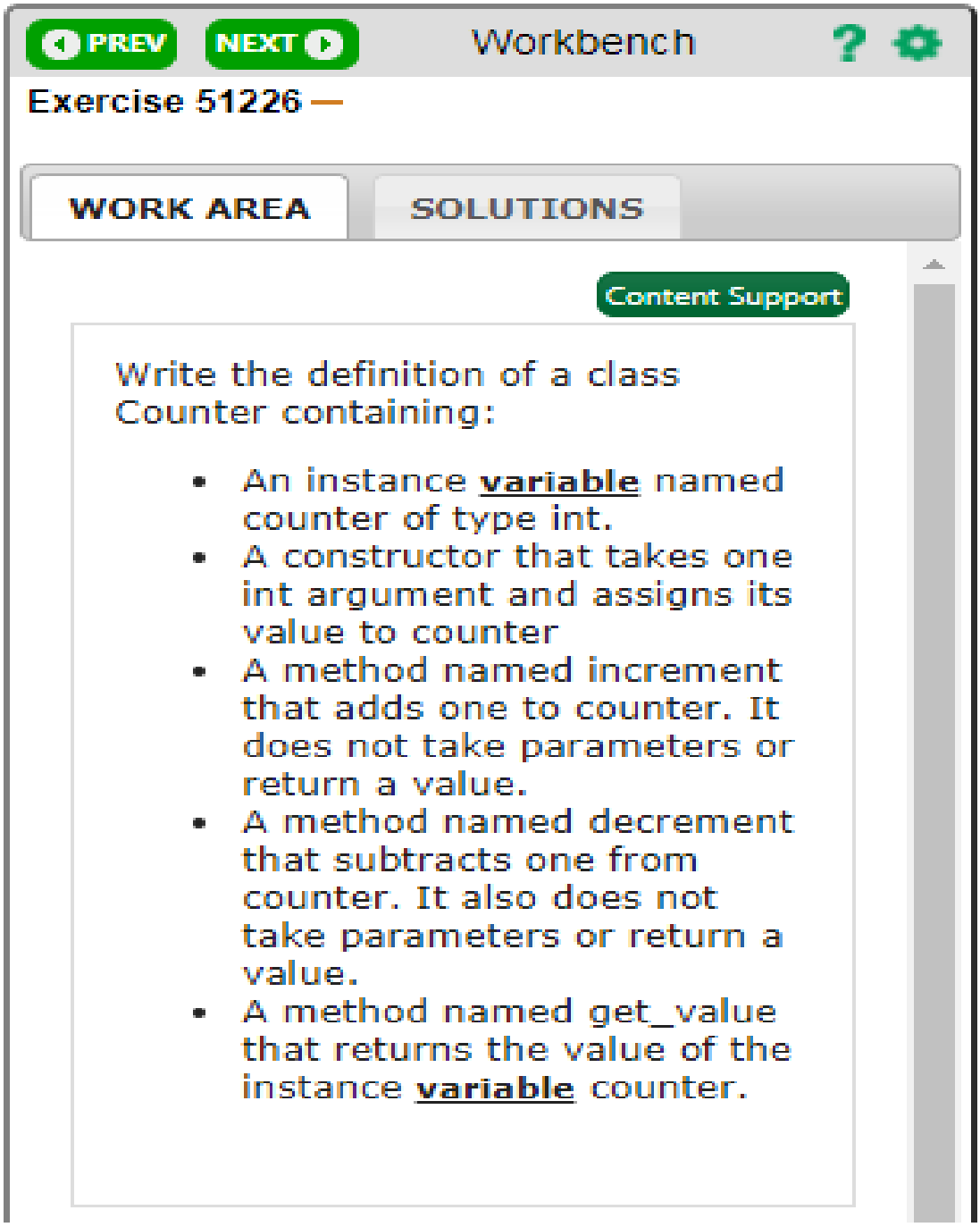
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

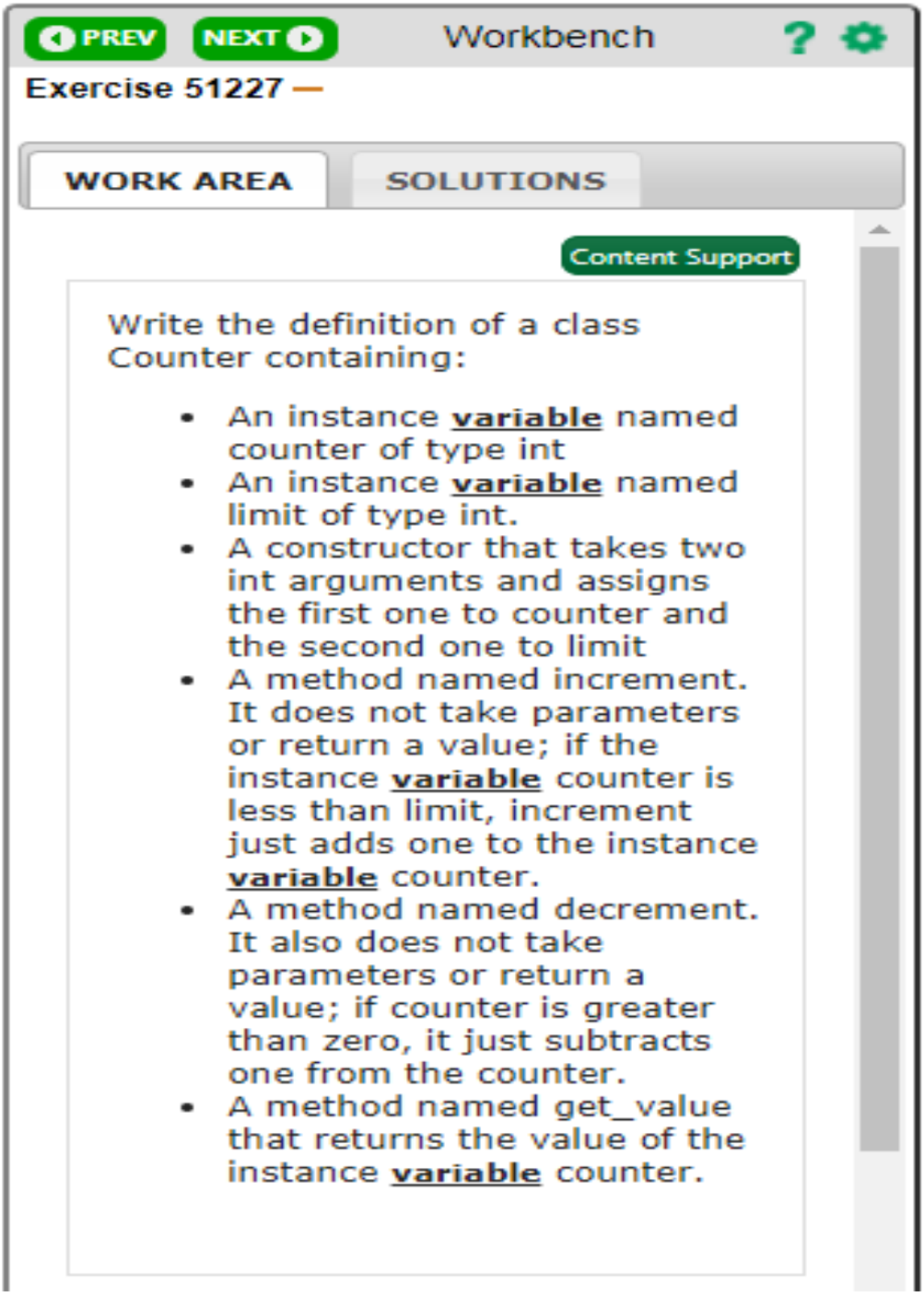
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

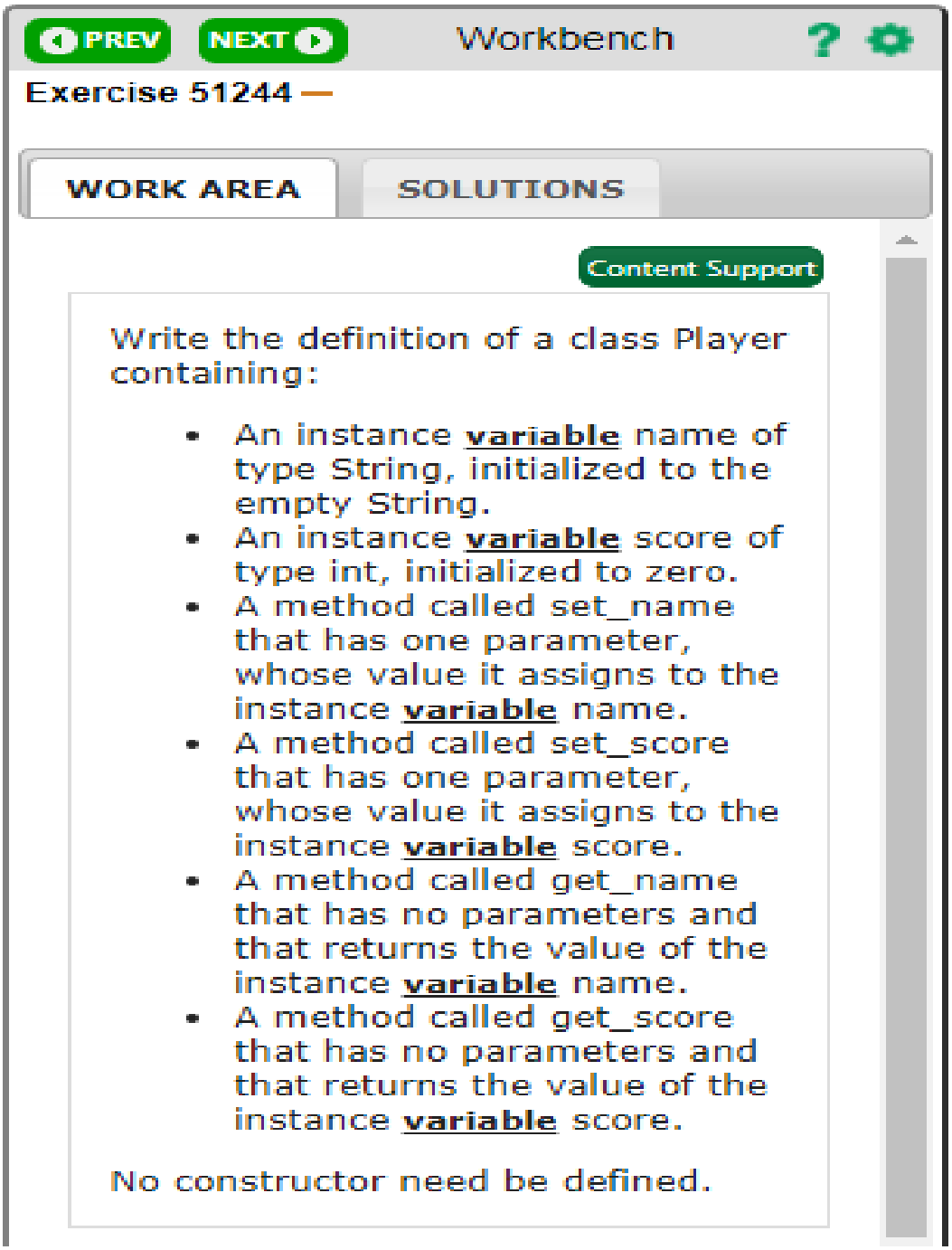
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

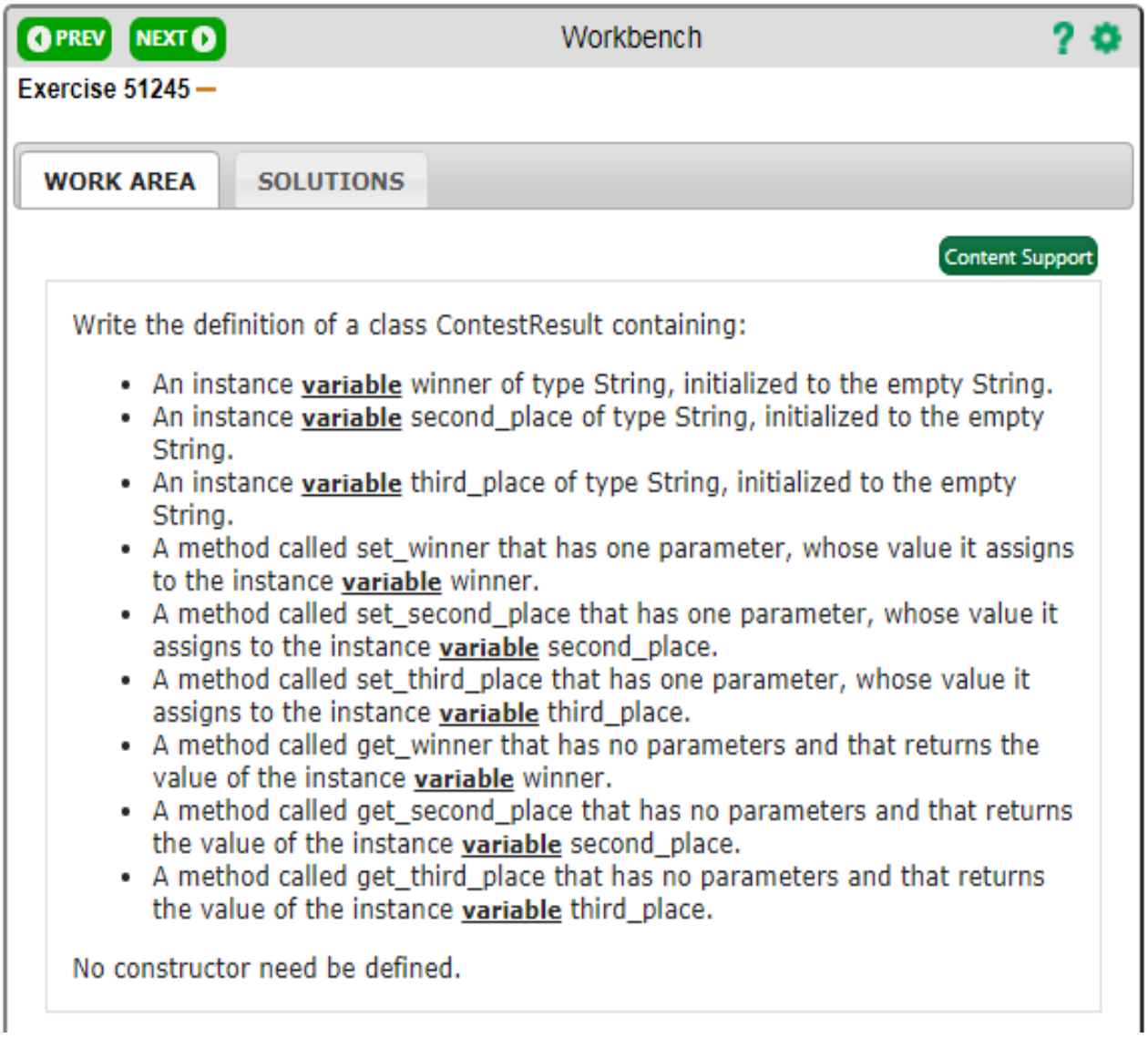
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

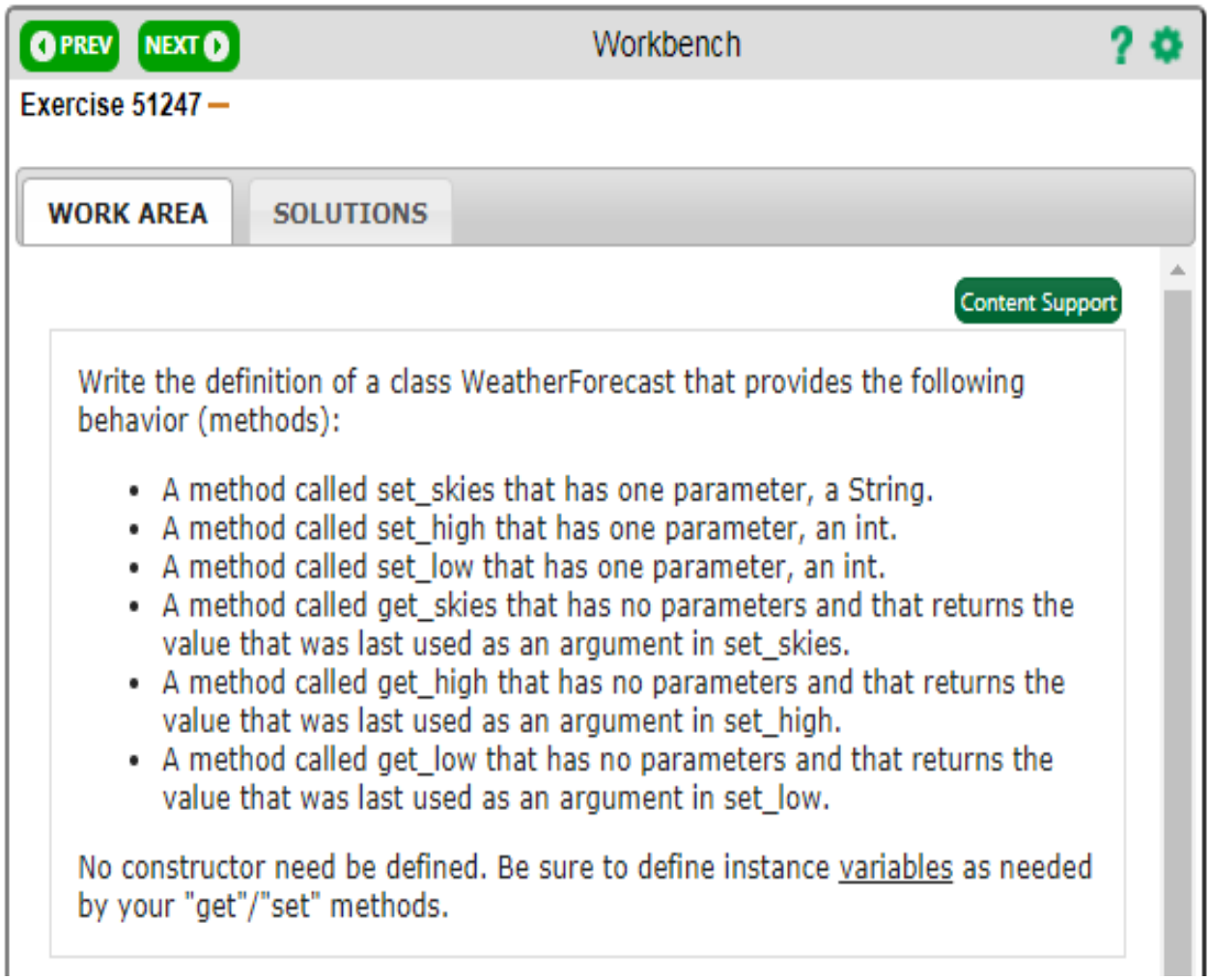
Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank

Programming Exercise



Programming Exercise Worksheet

this page is intentionally left blank