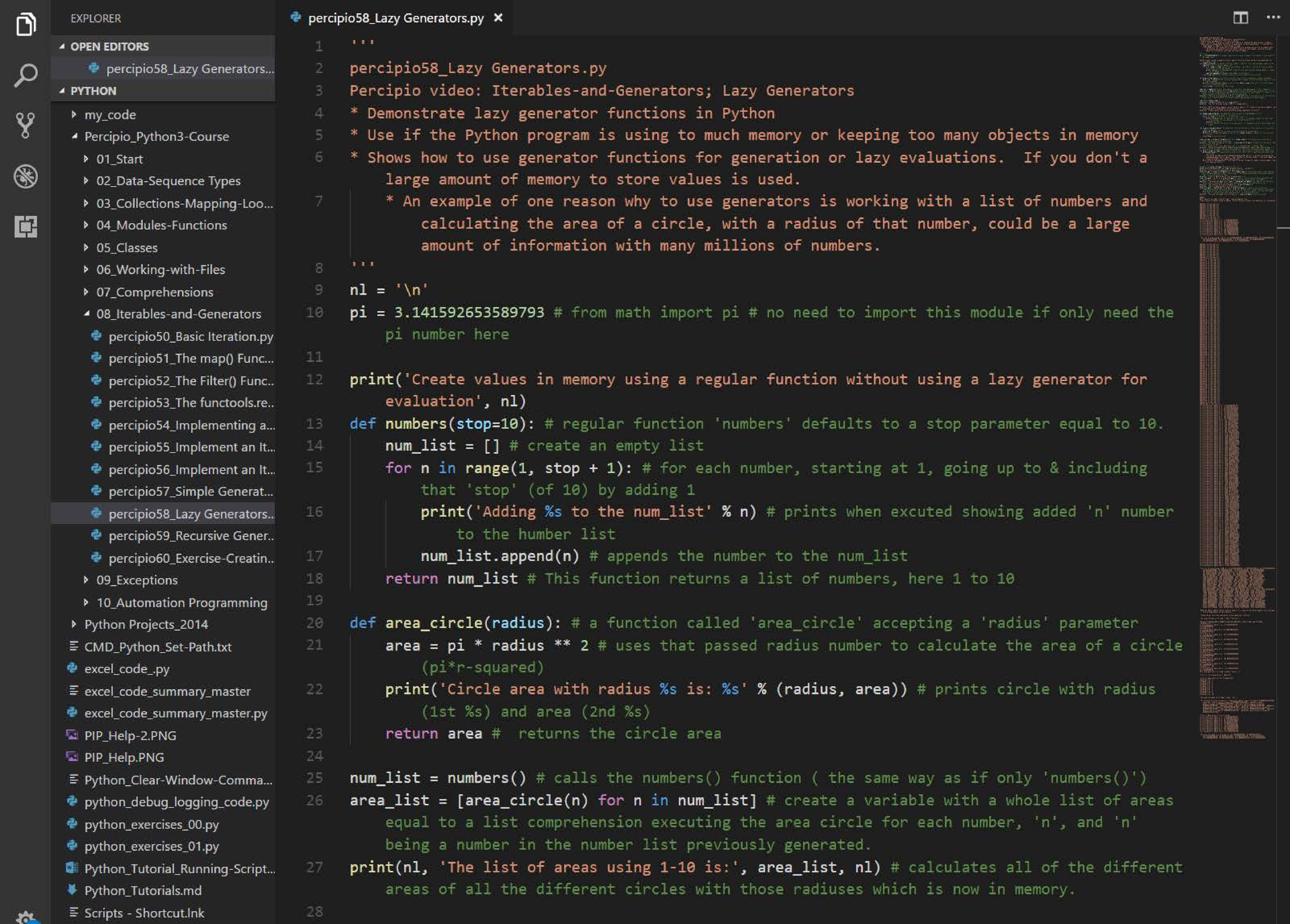
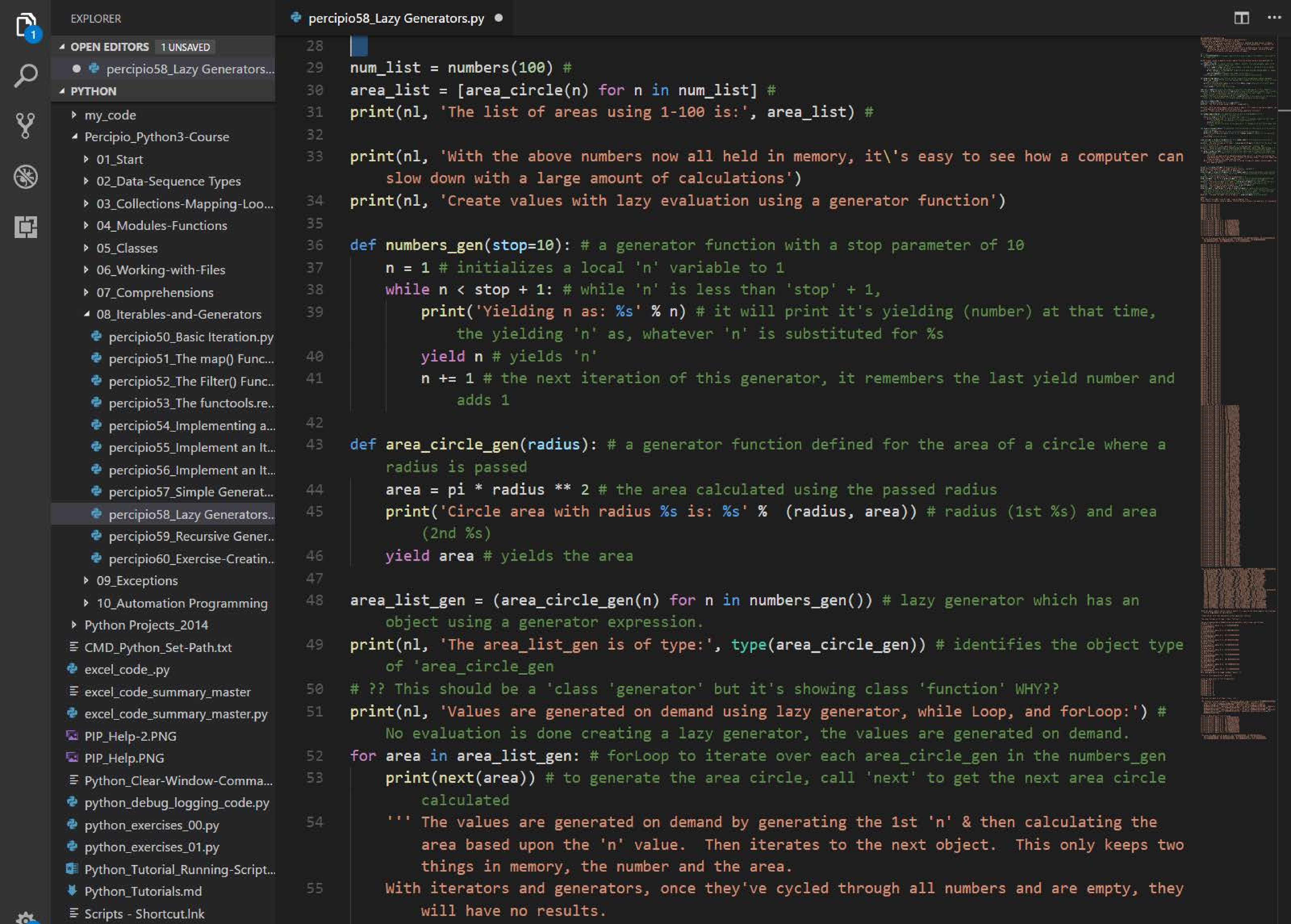
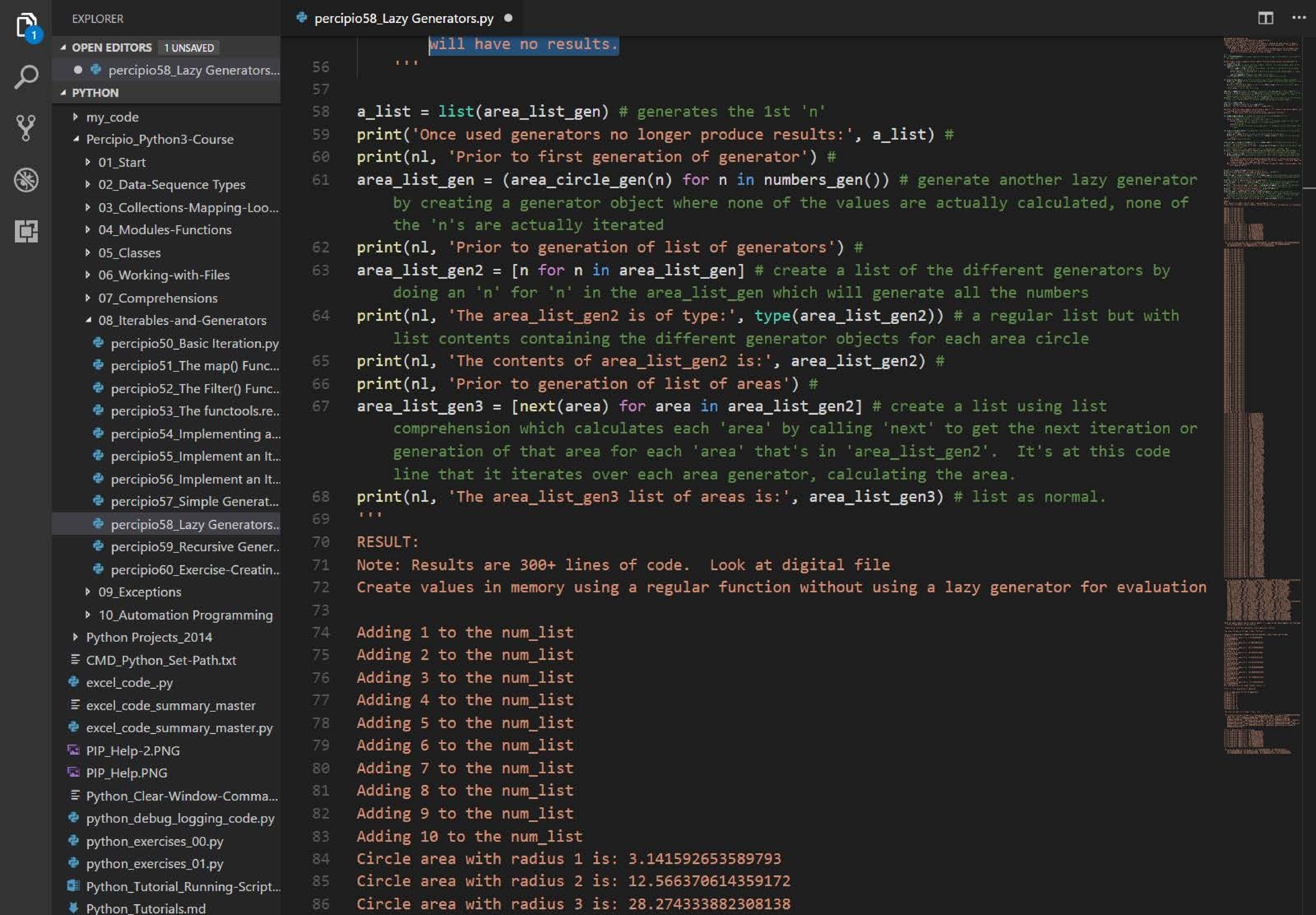

EXPLORER
OPEN EDITORS
percipio58_Lazy Generators...
PYTHON
my_code
Percipio_Python3-Course
01_Start
02_Data-Sequence Types
03_Collections-Mapping-Loo...
04_Modules-Functions
05_Classes
06_Working-with-Files
07_Comprehensions
08_Iterables-and-Generators
percipio50_Basic Iteration.py
percipio51_The map() Func...
percipio52_The Filter() Func...
percipio53_The functools.re...
percipio54_Implementing a...
percipio55_Implement an It...
percipio56_Implement an It...
percipio57_Simple Generat...
percipio58_Lazy Generators...
percipio59_Recursive Gener...
percipio60_Exercise-Creatin...
09_Exceptions
10_Automation Programming
Python Projects_2014
CMD_Python_Set-Path.txt
excel_code_.py
excel_code_summary_master
excel_code_summary_master.py
PIP_Help-2.PNG
PIP_Help.PNG
Python_Clear-Window-Comma...
python_debug_logging_code.py
python_exercises_00.py
python_exercises_01.py
Python_Tutorial_Running-Script...
Python_Tutorials.md
Scripts - Shortcut.lnk
percipio58_Lazy Generators.py

```python
'''
percipio58_Lazy Generators.py
Percipio video: Iterables-and-Generators; Lazy Generators
* Demonstrate lazy generator functions in Python
* Use if the Python program is using to much memory or keeping too many objects in memory
* Shows how to use generator functions for generation or lazy evaluations.  If you don't a large amount of memory to store values is used.
    * An example of one reason why to use generators is working with a list of numbers and calculating the area of a circle, with a radius of that number, could be a large amount of information with many millions of numbers.
'''
nl = '\n'
pi = 3.141592653589793 # from math import pi # no need to import this module if only need the pi number here

print('Create values in memory using a regular function without using a lazy generator for evaluation', nl)
def numbers(stop=10): # regular function 'numbers' defaults to a stop parameter equal to 10.
    num_list = [] # create an empty list
    for n in range(1, stop + 1): # for each number, starting at 1, going up to & including that 'stop' (of 10) by adding 1
        print('Adding %s to the num_list' % n) # prints when excuted showing added 'n' number to the humber list
        num_list.append(n) # appends the number to the num_list
    return num_list # This function returns a list of numbers, here 1 to 10

def area_circle(radius): # a function called 'area_circle' accepting a 'radius' parameter
    area = pi * radius ** 2 # uses that passed radius number to calculate the area of a circle (pi*r-squared)
    print('Circle area with radius %s is: %s' % (radius, area)) # prints circle with radius (1st %s) and area (2nd %s)
    return area #  returns the circle area

num_list = numbers() # calls the numbers() function ( the same way as if only 'numbers()')
area_list = [area_circle(n) for n in num_list] # create a variable with a whole list of areas equal to a list comprehension executing the area circle for each number, 'n', and 'n' being a number in the number list previously generated.
print(nl, 'The list of areas using 1-10 is:', area_list, nl) # calculates all of the different areas of all the different circles with those radiuses which is now in memory.

```


EXPLORER
percipio58_Lazy Generators.py
OPEN EDITORS 1 UNSAVED
percipio58_Lazy Generators...
PYTHON
my_code
Percipio_Python3-Course
01_Start
02_Data-Sequence Types
03_Collections-Mapping-Loo...
04_Modules-Functions
05_Classes
06_Working-with-Files
07_Comprehensions
08_Iterables-and-Generators
percipio50_Basic Iteration.py
percipio51_The map() Func...
percipio52_The Filter() Func..
percipio53_The functools.re..
percipio54_Implementing a...
percipio55_Implement an It...
percipio56_Implement an It...
percipio57_Simple Generat...
percipio58_Lazy Generators...
percipio59_Recursive Gener..
percipio60_Exercise-Creatin...
09_Exceptions
10_Automation Programming
Python Projects_2014
CMD_Python_Set-Path.txt
excel_code_.py
excel_code_summary_master
excel_code_summary_master.py
PIP_Help-2.PNG
PIP_Help.PNG
Python_Clear-Window-Comma...
python_debug_logging_code.py
python_exercises_00.py
python_exercises_01.py
Python_Tutorial_Running-Script...
Python_Tutorials.md
Scripts - Shortcut.lnk

```python

num_list = numbers(100) #
area_list = [area_circle(n) for n in num_list] #
print(nl, 'The list of areas using 1-100 is:', area_list) #

print(nl, 'With the above numbers now all held in memory, it\'s easy to see how a computer can slow down with a large amount of calculations')
print(nl, 'Create values with lazy evaluation using a generator function')

def numbers_gen(stop=10): # a generator function with a stop parameter of 10
    n = 1 # initializes a local 'n' variable to 1
    while n < stop + 1: # while 'n' is less than 'stop' + 1,
        print('Yielding n as: %s' % n) # it will print it's yielding (number) at that time, the yielding 'n' as, whatever 'n' is substituted for %s
        yield n # yields 'n'
        n += 1 # the next iteration of this generator, it remembers the last yield number and adds 1

def area_circle_gen(radius): # a generator function defined for the area of a circle where a radius is passed
    area = pi * radius ** 2 # the area calculated using the passed radius
    print('Circle area with radius %s is: %s' %  (radius, area)) # radius (1st %s) and area (2nd %s)
    yield area # yields the area

area_list_gen = (area_circle_gen(n) for n in numbers_gen()) # lazy generator which has an object using a generator expression.
print(nl, 'The area_list_gen is of type:', type(area_circle_gen)) # identifies the object type of 'area_circle_gen
# ?? This should be a 'class 'generator' but it's showing class 'function' WHY??
print(nl, 'Values are generated on demand using lazy generator, while Loop, and forLoop:') # No evaluation is done creating a lazy generator, the values are generated on demand.
for area in area_list_gen: # forLoop to iterate over each area_circle_gen in the numbers_gen
    print(next(area)) # to generate the area circle, call 'next' to get the next area circle calculated
    ''' The values are generated on demand by generating the 1st 'n' & then calculating the area based upon the 'n' value.  Then iterates to the next object.  This only keeps two things in memory, the number and the area.
    With iterators and generators, once they've cycled through all numbers and are empty, they will have no results.
```


EXPLORER
OPEN EDITORS 1 UNSAVED
percipio58_Lazy Generators...
PYTHON
my_code
Percipio_Python3-Course
01_Start
02_Data-Sequence Types
03_Collections-Mapping-Loo...
04_Modules-Functions
05_Classes
06_Working-with-Files
07_Comprehensions
08_Iterables-and-Generators
percipio50_Basic Iteration.py
percipio51_The map() Func...
percipio52_The Filter() Func...
percipio53_The functools.re...
percipio54_Implementing a...
percipio55_Implement an It...
percipio56_Implement an It...
percipio57_Simple Generat...
percipio58_Lazy Generators...
percipio59_Recursive Gener...
percipio60_Exercise-Creatin...
09_Exceptions
10_Automation Programming
Python Projects_2014
CMD_Python_Set-Path.txt
excel_code_.py
excel_code_summary_master
excel_code_summary_master.py
PIP_Help-2.PNG
PIP_Help.PNG
Python_Clear-Window-Comma...
python_debug_logging_code.py
python_exercises_00.py
python_exercises_01.py
Python_Tutorial_Running-Script...
Python_Tutorials.md
percipio58_Lazy Generators.py

```
    will have no results.
    '''

a_list = list(area_list_gen) # generates the 1st 'n'
print('Once used generators no longer produce results:', a_list) #
print(nl, 'Prior to first generation of generator') #
area_list_gen = (area_circle_gen(n) for n in numbers_gen()) # generate another lazy generator
    by creating a generator object where none of the values are actually calculated, none of
    the 'n's are actually iterated
print(nl, 'Prior to generation of list of generators') #
area_list_gen2 = [n for n in area_list_gen] # create a list of the different generators by
    doing an 'n' for 'n' in the area_list_gen which will generate all the numbers
print(nl, 'The area_list_gen2 is of type:', type(area_list_gen2)) # a regular list but with
    list contents containing the different generator objects for each area circle
print(nl, 'The contents of area_list_gen2 is:', area_list_gen2) #
print(nl, 'Prior to generation of list of areas') #
area_list_gen3 = [next(area) for area in area_list_gen2] # create a list using list
    comprehension which calculates each 'area' by calling 'next' to get the next iteration or
    generation of that area for each 'area' that's in 'area_list_gen2'.  It's at this code
    line that it iterates over each area generator, calculating the area.
print(nl, 'The area_list_gen3 list of areas is:', area_list_gen3) # list as normal.
'''
RESULT:
Note: Results are 300+ lines of code.  Look at digital file
Create values in memory using a regular function without using a lazy generator for evaluation

Adding 1 to the num_list
Adding 2 to the num_list
Adding 3 to the num_list
Adding 4 to the num_list
Adding 5 to the num_list
Adding 6 to the num_list
Adding 7 to the num_list
Adding 8 to the num_list
Adding 9 to the num_list
Adding 10 to the num_list
Circle area with radius 1 is: 3.141592653589793
Circle area with radius 2 is: 12.566370614359172
Circle area with radius 3 is: 28.274333882308138
```