Object-Oriented Programming

1.

a. class Country:

    def __init__(self, name, population, area):
        """
        (Country, str, int, int)
        A new Country named name with population people and area area.
        """
        self.name = name
        self.population = population
        self.area = area

b. def is_larger(self, other):
    """
    (Country, Country) -> bool
    Return whether this country is larger than other.
    """
    return self.area > other.area

c. def population_density(self):
    """
    (Country) -> float
    Return the population density of this country.
    """
    return self.population / self.area
return self.population / self.area

d.
def __str__(self):
    """ (Country) -> str

    Return a printable representation of this country.

    >>> usa = Country('United States of America', 313914040, 9826675)
    >>> print(usa)
    United States of America has a population of 313914040 and is 9826675 square km.
    """

    return '{} has a population of {} and is {} square km.'
    .format(self.name, self.population, self.area)

e.
def __repr__(self):
    """ (Country) -> str

    Return a concise representation of this country.

    >>> canada = Country('Canada', 34482779, 9984670)
    >>> canada
    Country('Canada', 34482779, 9984670)
    >>> [canada]
    "34482779, 9984670)"::http://pragprog.com/wikis/wiki/Country('Canada',
    ""

    return "Country('{0}', {1}, {2})".format(
        self.name, self.population, self.area)

2.
a.
class Continent:

def __init__(self, name, countries):
    """ (Continent, str, list of Country) -> NoneType

    A continent named name made up of countries.

    >>> canada = country.Country('Canada', 34482779, 9984670)
    >>> usa = country.Country('United States of America', 313914040, ...
        9826675)
    >>> mexico = country.Country('Mexico', 112336538, 1943950)
    >>> countries = [canada, usa, mexico]
    >>> north_america = Continent('North America', countries)
    >>> north_america.name
'North America'
>>> for country in north_america.countries:
...     print(country)
Canada has a population of 34482779 and is 9984670 square km.
United States of America has a population of 313914040 and is 9826675
square km.
Mexico has a population of 112336538 and is 1943950 square km.

self.name = name
self.countries = countries

b.

def total_population(self):
    """ (Continent) -> int

    Return the total population of all the
countries in this continent.
    """

>>> canada = country.Country('Canada', 34482779, 9984670)
>>> usa = country.Country('United States of America', 313914040,
... 9826675)
>>> mexico = country.Country('Mexico', 112336538, 1943950)
>>> countries = [canada, usa, mexico]
>>> north_america = Continent('North America', countries)
>>> north_america.total_population
460733357

""

total = 0
for country in self.countries:
    total = total + country.population
return total

c.

def __str__(self):
    """ (Continent) -> str

    Return a printable representation of this Continent.
    """

>>> canada = country.Country('Canada', 34482779, 9984670)
>>> usa = country.Country('United States of America', 313914040,
... 9826675)
>>> mexico = country.Country('Mexico', 112336538, 1943950)
>>> countries = [canada, usa, mexico]
>>> north_america = Continent('North America', countries)
>>> print(north_america)
North America
Canada has a population of 34482779 and is 9984670 square km.
United States of America has a population of 313914040 and is 9826675
square km.
Mexico has a population of 112336538 and is 1943950 square km.
res = self.name
for country in self.countries:
    res = res + ',
' + str(country)

return res

3.
a.
def __str__(self):
    """ (Student) -> str
    Return a string representation of this Student.
    >>> student = Student('Paul', 'Ajax', 'pgries@cs.toronto.edu', '1234')
    >>> student.__str__()
    'Paul
Ajax
pgries@cs.toronto.edu
1234
Previous courses: '

    """
    member_string = super().__str__()
    return '''{}\n{}\nPrevious courses: {}
Current courses: {}'''.format(
        member_string,
        self.student_number,
        self.courses_taken,
        self.courses_taking)

b.

# Member repr:
def __repr__(self):
    """ (Member) -> str
    Return a concise string representation of this Member.
    >>> member = Member('Paul', 'Ajax', 'pgries@cs.toronto.edu')
    >>> member.__repr__()
    "Member('Paul', 'Ajax', 'pgries@cs.toronto.edu')"

    """
    return "Member('{}', '{}', '{}').format(
        self.name, self.address, self.email)

# Faculty repr:
def __repr__(self):
    """ (Faculty) -> str
Return a concise string representation of this Faculty.

```python
>>> faculty = Faculty('Paul', 'Ajax', 'pgries@cs.toronto.edu', '1234')
```

```python
>>> faculty.__repr__()
"Faculty('Paul', 'Ajax', 'pgries@cs.toronto.edu', 1234, [])"
```

```python
"Faculty('{}', '{}', '{}', {}, [{}, {}, {}, {}]).format(
    self.name, self.address, self.email, self.faculty_number,
    ', ', ').join(self.courses_teaching))
```

# Student repr:

```python
    def __repr__(self):
        """ (Faculty) -> str
        Return a concise string representation of this Faculty.
        >>> student = Student('Paul', 'Ajax', 'pgries@cs.toronto.edu', '1234')
        >>> student.__repr__()
        "Student('Paul', 'Ajax', 'pgries@cs.toronto.edu', 1234, [], [], [])"
        ""
        return "Student('{}', '{}', '{}', {}, [{}, {}, {}, {}, {}]).format(
            self.name, self.address, self.email, self.student_number,
            ', ', ').join(self.courses_taking)
```

4.

```python
class Nematode:
    """ A microscopic worm. """
    def __init__(self, length, gender, age):
        """ (Nematode, float, str, int) -> NoneType
        Create a new Nematode with body length (in millimeters; they are
        about 1 mm in length), gender (either hermaphrodite or male), and age (in
days).
        >>> worm = Nematode(1.1, 'hermaphrodite', 2)
        >>> worm.length
        1.1
        >>> worm.gender
        'hermaphrodite'
        >>> worm.age
        2
        ""
        self.length = length
        self.gender = gender
        self.age = age
    def __str__(self):
        """
Return a string representation of this Nematode.

```python
>>> worm = Nematode(1.1, 'hermaphrodite', 2)
>>> worm.__str__()
'Nematode: 1.1mm long, gender is hermaphrodite, 2 days old'
```

```python
return 'Nematode: {}mm long, gender is {}, {} days old'.format(self.length, self.gender, self.age)
def __repr__(self):
    """ (Nematode) -> str
    Return a concise string representation of this Nematode.
    ""
    return "Nematode({}, '{}', {})".format(self.length, self.gender, self.age)
```

5.

a.

class Point:
    def __init__(self, x, y):
        """ (Point, int, int) -> NoneType
        A new Point at position (x, y).
        ""
        self.x = x
        self.y = y

b.

class LineSegment:
    def __init__(self, point1, point2):
        """ (LineSegment, Point, Point) -> NoneType
        A new LineSegment connecting point1 to point2.
        ""
        >>> p1 = Point(1, 3)
        >>> p2 = Point(3, 2)
>>> segment = LineSegment(p1, p2)
>>> segment.startpoint == p1
True
>>> segment.endpoint == p2
True

self.startpoint = point1
self.endpoint = point2

c.

def slope(self):
    """ (LineSegment) -> float

    >>> segment = LineSegment(Point(1, 1), Point(3, 2))
    >>> segment.slope()
    0.5
    """

    return (self.endpoint.y - self.startpoint.y) / \n           (self.endpoint.x - self.startpoint.x)

d.

def length(self):
    """ (LineSegment) -> float

    >>> segment = LineSegment(Point(1, 1), Point(3, 2))
    >>> segment.length()
    2.23606797749979
    """

    return math.sqrt(
        (self.endpoint.x - self.startpoint.x) ** 2 +
        (self.endpoint.y - self.startpoint.y) ** 2)