```
1   import check
2   import math
3
4
5   packing_density = 0.64
6
7   def gumball_approximation(jar_radius, jar_height, gumball_radius):
8     '''
9     Returns an approximation of how many gumballs are in a cylindrical
10    jar given the jar_radius, jar_height and the gumball_radius
11
12    gumball_approximation: Float Float Float -> Nat
13    Requires:
14       0.0 < jar_radius, jar_height, gumball_radius
15
16    Examples:
17       gumball_approximation(10.0, 10.0, 1.0) => 480
18    '''
19    volume_of_jar = math.pi * jar_radius**2 * jar_height
20    volume_of_gumball = 4/3 * math.pi * gumball_radius**3
21    return round((packing_density * volume_of_jar)/volume_of_gumball)
22
23  ##Examples:
24  check.expect("Ex1", gumball_approximation(10.0, 10.0, 1.0) , 480)
25
26
27  ##Tests
28  check.expect("Minimum", gumball_approximation(1.0, 2.0, 1.0) , 1)
29  check.expect("Random", gumball_approximation(432.0, 123.0, 55.0) , 66)
30  check.expect("Unrealistic", gumball_approximation(4320.0, 1230.0, 0.3),
31               408084480000)
32  check.expect("plausible", gumball_approximation(50.0, 10.0, 5.0), 96)
```

```
 1  import check
 2
 3  def sum_digits_two_digit_number(n):
 4    '''
 5    Returns the sum of the digits of a two digit number
 6
 7    sum_digits_two_digit_number: Nat -> Nat
 8    Requires: 0 <= n <= 99
 9    '''
10    return n % 10 + n // 10
11
12  def lshift(n, exp):
13    '''
14    Returns the (n+1)st digit of a number where
15    the first digit is the rightmost digit and so on.
16
17    lshift: Nat Nat -> Nat
18    '''
19    return (n // 10**exp) % 10
20
21  def make_sin(partial_number):
22    '''
23    Returns the valid SIN number beginning with
24    the 8 digit partial_number.
25
26    make_sin: Nat -> Nat
27    Requires: 0 <= partial_number < 10**9
28    (also okay if 10**8 <= partial_number < 10**9)
29
30    Examples:
31       make_sin(10000000) => 100000009
32       make_sin(64075429) => 640754297
33       make_sin(99999999) => 999999998
34    '''
35    d8 = lshift(partial_number, 0)
36    d7 = lshift(partial_number, 1)
37    d6 = lshift(partial_number, 2)
38    d5 = lshift(partial_number, 3)
39    d4 = lshift(partial_number, 4)
40    d3 = lshift(partial_number, 5)
```

```
41      d2 = lshift(partial_number, 6)
42      d1 = lshift(partial_number, 7)
43      o = d1 + d3 + d5 + d7
44      e = sum_digits_two_digit_number(2*d2) + \
45          sum_digits_two_digit_number(2*d4) + \
46          sum_digits_two_digit_number(2*d6) + \
47          sum_digits_two_digit_number(2*d8)
48      check_sum = (10 - (e + o) % 10) % 10
49      return partial_number * 10 + check_sum
50
51  ##Examples:
52  check.expect("Test Lower Edge",  make_sin(10000000), 100000009)
53  check.expect("Test given",  make_sin(64075429), 640754297)
54  check.expect("Test Upper Edge",  make_sin(99999999), 999999998)
55
56  ##Tests:
57  check.expect("Test 8",  make_sin(10000010), 100000108)
58  check.expect("Test 7",  make_sin(10000020), 100000207)
59  check.expect("Test 6",  make_sin(10000030), 100000306)
60  check.expect("Test 5",  make_sin(10000040), 100000405)
61  check.expect("Test 4",  make_sin(10000050), 100000504)
62  check.expect("Test 3",  make_sin(10000060), 100000603)
63  check.expect("Test 2",  make_sin(10000070), 100000702)
64  check.expect("Test 1",  make_sin(10000080), 100000801)
65  check.expect("Test 0",  make_sin(10000090), 100000900)
66  check.expect("Test random",  make_sin(23874663), 238746630)
67
```

```
1   import check
2   import math
3
4
5   ##We reuse some of the helper functions in the math module lesson!
6
7
8   def sector_area(r, theta):
9     '''
10    Returns the area of a sector of a circle
11    given the radius r and angle theta
12
13    sector_area: Float Float  -> Float
14    Requires:
15       0.0 <= r
16       0.0 <= theta <= 2*math.pi
17    '''
18    return 1/2 * r**2 * theta
19
20  def triangle_area(base, height):
21    '''
22    Returns the area of a triangle given the base and height
23
24    sector_area: Float Float  -> Float
25    Requires:
26       0.0 <= base, height
27    '''
28    return 1/2 * base * height
29
30  def shaded_region(r, d):
31    '''
32    Returns the common area between two overlapping circles
33    where d is the distance between centres and r is the radii of
34    the circles
35
36    shaded_region: Float Float -> Float
37    Requires:0.0 < d < 2r
38
39    Example:
40       shaded_region(2.0, 1.0) => 8.608436900118837
```

```
41      '''
42      ## Notice that connecting the centres and intersection
43      ## points creates four identical triangles
44      theta = math.acos(d/(2*r))
45      distance_between_intersection_points = 2*r*math.sin(theta)
46      half_shaded = (sector_area(r, 2*theta) -
47                     triangle_area(distance_between_intersection_points,
48                                   d/2))
49      area = 2 * half_shaded
50      return area
51
52
53  EPSILON = 0.00001
54
55  ##Example:
56  check.within("Example", shaded_region(2.0, 1.0),
57               8.608436900118837, EPSILON)
58
59  ##Tests
60  check.within("d == r", shaded_region(1.0, 1.0),
61               1.2283696986087567, EPSILON)
62  check.within("large", shaded_region(555.3, 123.4),
63               831969.9865943828, EPSILON)
64  check.within("tiny", shaded_region(0.0066, 0.0005),
65               0.00013024935461305247, EPSILON)
```