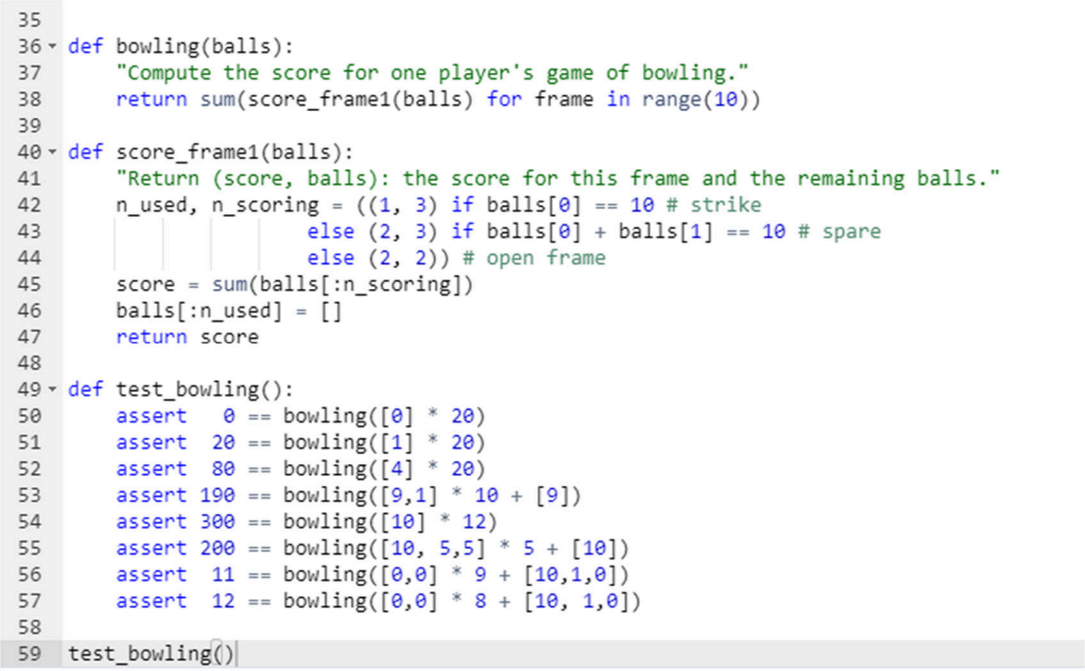
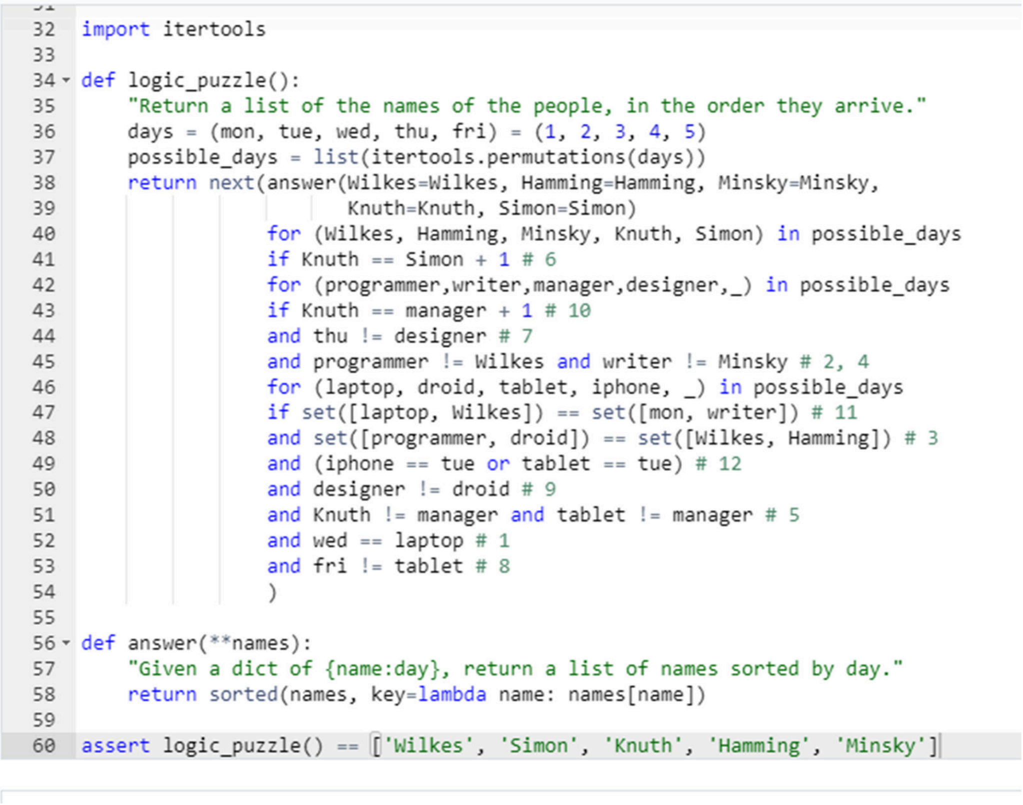
Bowling



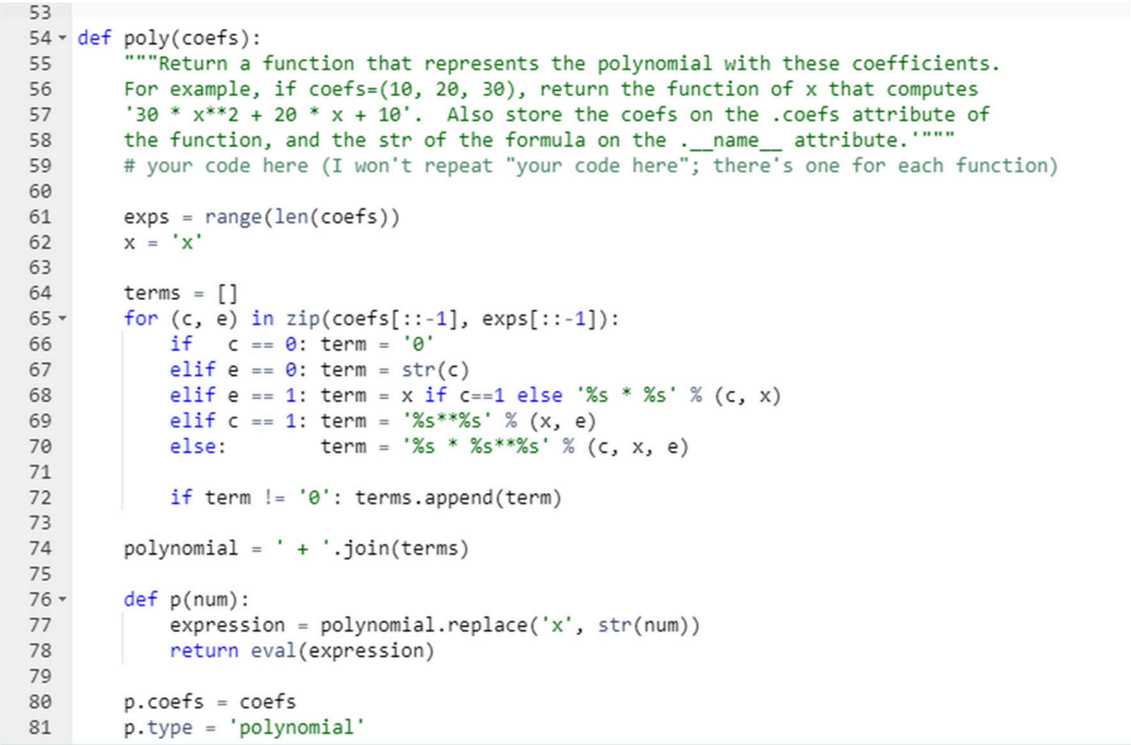
```python
def bowling(balls):
    "Compute the score for one player's game of bowling."
    return sum(score_frame1(balls) for frame in range(10))

def score_frame1(balls):
    "Return (score, balls): the score for this frame and the remaining balls."
    n_used, n_scoring = ((1, 3) if balls[0] == 10 # strike
                    else (2, 3) if balls[0] + balls[1] == 10 # spare
                    else (2, 2)) # open frame
    score = sum(balls[:n_scoring])
    balls[:n_used] = []
    return score

def test_bowling():
    assert   0 == bowling([0] * 20)
    assert  20 == bowling([1] * 20)
    assert  80 == bowling([4] * 20)
    assert 190 == bowling([9,1] * 10 + [9])
    assert 300 == bowling([10] * 12)
    assert 200 == bowling([10, 5,5] * 5 + [10])
    assert  11 == bowling([0,0] * 9 + [10,1,0])
    assert  12 == bowling([0,0] * 8 + [10, 1,0])

test_bowling()
```

Logic Puzzle



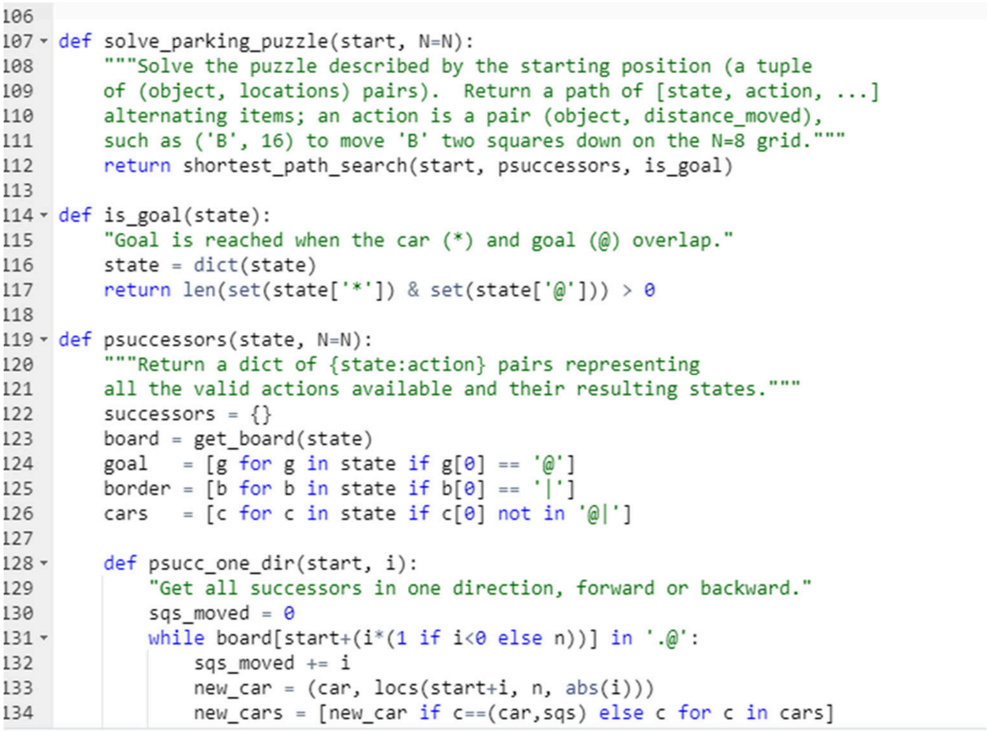
```python
import itertools

def logic_puzzle():
    "Return a list of the names of the people, in the order they arrive."
    days = (mon, tue, wed, thu, fri) = (1, 2, 3, 4, 5)
    possible_days = list(itertools.permutations(days))
    return next(answer(Wilkes=Wilkes, Hamming=Hamming, Minsky=Minsky,
                       Knuth=Knuth, Simon=Simon)
                for (Wilkes, Hamming, Minsky, Knuth, Simon) in possible_days
                if Knuth == Simon + 1 # 6
                for (programmer,writer,manager,designer,_) in possible_days
                if Knuth == manager + 1 # 10
                and thu != designer # 7
                and programmer != Wilkes and writer != Minsky # 2, 4
                for (laptop, droid, tablet, iphone, _) in possible_days
                if set([laptop, Wilkes]) == set([mon, writer]) # 11
                and set([programmer, droid]) == set([Wilkes, Hamming]) # 3
                and (iphone == tue or tablet == tue) # 12
                and designer != droid # 9
                and Knuth != manager and tablet != manager # 5
                and wed == laptop # 1
                and fri != tablet # 8
                )

def answer(**names):
    "Given a dict of {name:day}, return a list of names sorted by day."
    return sorted(names, key=lambda name: names[name])

assert logic_puzzle() == ['Wilkes', 'Simon', 'Knuth', 'Hamming', 'Minsky']
```

Polynomials



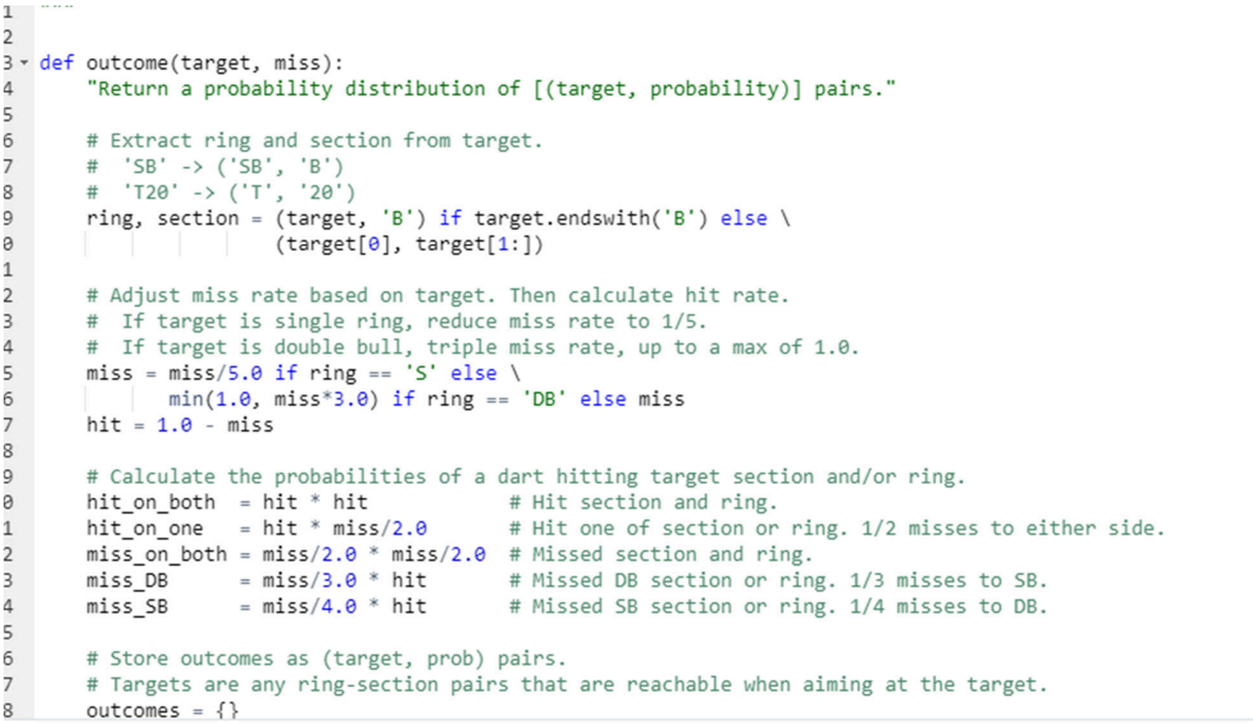
```python
def poly(coefs):
    """Return a function that represents the polynomial with these coefficients.
    For example, if coefs=(10, 20, 30), return the function of x that computes
    '30 * x**2 + 20 * x + 10'.  Also store the coefs on the .coefs attribute of
    the function, and the str of the formula on the .__name__ attribute.'"""
    # your code here (I won't repeat "your code here"; there's one for each function)

    exps = range(len(coefs))
    x = 'x'

    terms = []
    for (c, e) in zip(coefs[::-1], exps[::-1]):
        if   c == 0: term = '0'
        elif e == 0: term = str(c)
        elif e == 1: term = x if c==1 else '%s * %s' % (c, x)
        elif c == 1: term = '%s**%s' % (x, e)
        else:        term = '%s * %s**%s' % (c, x, e)

        if term != '0': terms.append(term)

    polynomial = ' + '.join(terms)

    def p(num):
        expression = polynomial.replace('x', str(num))
        return eval(expression)

    p.coefs = coefs
    p.type = 'polynomial'
```

Parking Lot Search



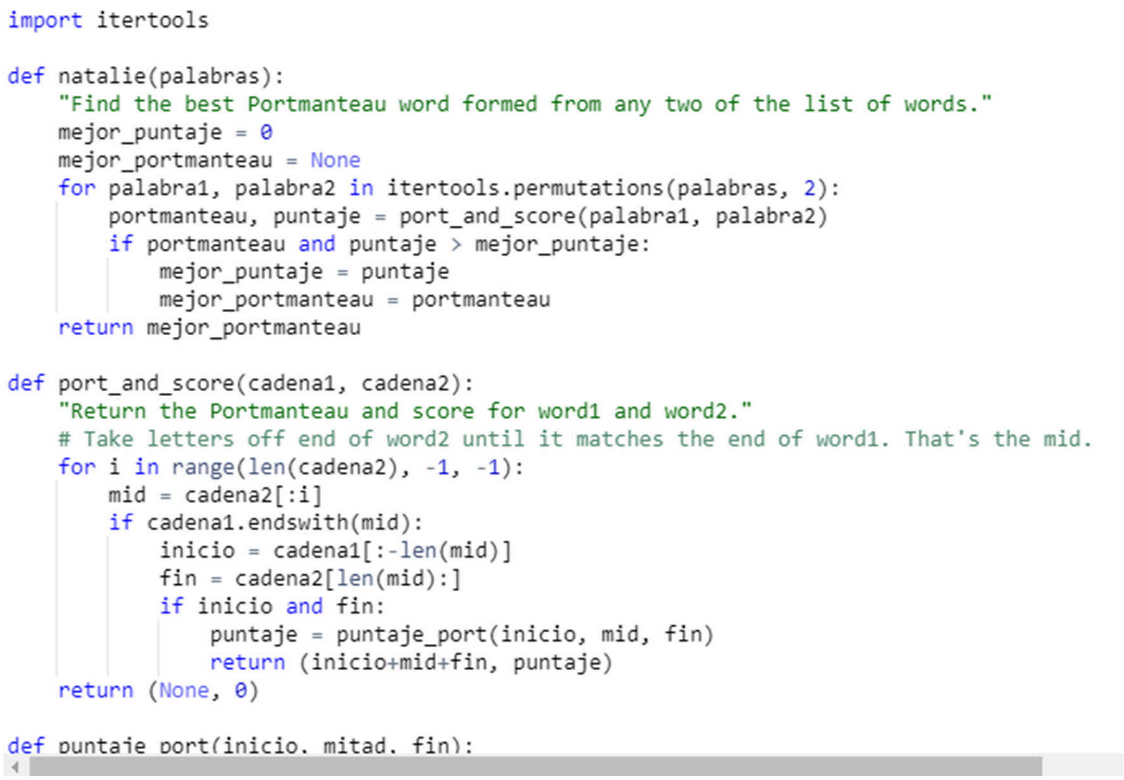
```python
def solve_parking_puzzle(start, N=N):
    """Solve the puzzle described by the starting position (a tuple
    of (object, locations) pairs).  Return a path of [state, action, ...]
    alternating items; an action is a pair (object, distance_moved),
    such as ('B', 16) to move 'B' two squares down on the N=8 grid."""
    return shortest_path_search(start, psuccessors, is_goal)

def is_goal(state):
    "Goal is reached when the car (*) and goal (@) overlap."
    state = dict(state)
    return len(set(state['*']) & set(state['@'])) > 0

def psuccessors(state, N=N):
    """Return a dict of {state:action} pairs representing
    all the valid actions available and their resulting states."""
    successors = {}
    board = get_board(state)
    goal   = [g for g in state if g[0] == '@']
    border = [b for b in state if b[0] == '|']
    cars   = [c for c in state if c[0] not in '@|']

    def psucc_one_dir(start, i):
        "Get all successors in one direction, forward or backward."
        sqs_moved = 0
        while board[start+(i*(1 if i<0 else n))] in '.@':
            sqs_moved += i
            new_car = (car, locs(start+i, n, abs(i)))
            new_cars = [new_car if c==(car,sqs) else c for c in cars]
```

Darts Probability



```python
"""


def outcome(target, miss):
    "Return a probability distribution of [(target, probability)] pairs."

    # Extract ring and section from target.
    #  'SB' -> ('SB', 'B')
    #  'T20' -> ('T', '20')
    ring, section = (target, 'B') if target.endswith('B') else \
                    (target[0], target[1:])

    # Adjust miss rate based on target. Then calculate hit rate.
    #  If target is single ring, reduce miss rate to 1/5.
    #  If target is double bull, triple miss rate, up to a max of 1.0.
    miss = miss/5.0 if ring == 'S' else \
           min(1.0, miss*3.0) if ring == 'DB' else miss
    hit = 1.0 - miss

    # Calculate the probabilities of a dart hitting target section and/or ring.
    hit_on_both  = hit * hit            # Hit section and ring.
    hit_on_one   = hit * miss/2.0       # Hit one of section or ring. 1/2 misses to either side.
    miss_on_both = miss/2.0 * miss/2.0  # Missed section and ring.
    miss_DB      = miss/3.0 * hit       # Missed DB section or ring. 1/3 misses to SB.
    miss_SB      = miss/4.0 * hit       # Missed SB section or ring. 1/4 misses to DB.

    # Store outcomes as (target, prob) pairs.
    # Targets are any ring-section pairs that are reachable when aiming at the target.
    outcomes = {}
```

Portmonteau



```python
import itertools

def natalie(palabras):
    "Find the best Portmanteau word formed from any two of the list of words."
    mejor_puntaje = 0
    mejor_portmanteau = None
    for palabra1, palabra2 in itertools.permutations(palabras, 2):
        portmanteau, puntaje = port_and_score(palabra1, palabra2)
        if portmanteau and puntaje > mejor_puntaje:
            mejor_puntaje = puntaje
            mejor_portmanteau = portmanteau
    return mejor_portmanteau

def port_and_score(cadena1, cadena2):
    "Return the Portmanteau and score for word1 and word2."
    # Take letters off end of word2 until it matches the end of word1. That's the mid.
    for i in range(len(cadena2), -1, -1):
        mid = cadena2[:i]
        if cadena1.endswith(mid):
            inicio = cadena1[:-len(mid)]
            fin = cadena2[len(mid):]
            if inicio and fin:
                puntaje = puntaje_port(inicio, mid, fin)
                return (inicio+mid+fin, puntaje)
    return (None, 0)

def puntaje_port(inicio, mitad, fin):
```