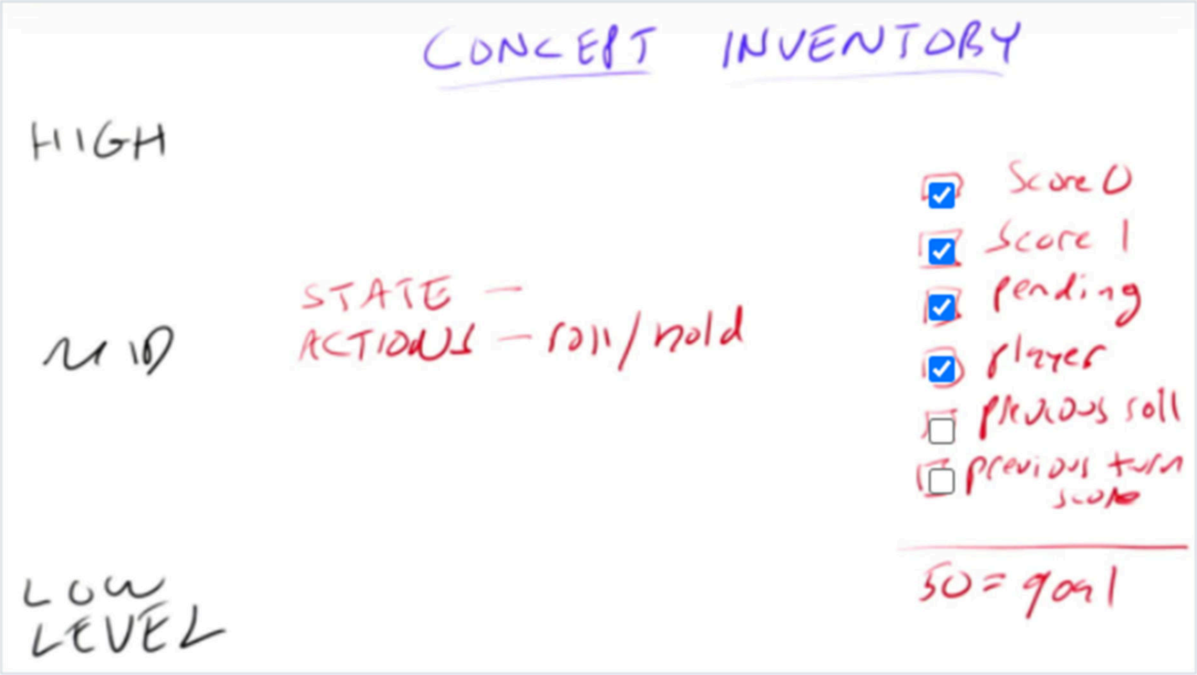
The State Of Pig



CONCEPT INVENTORY

HIGH

MID

STATE –

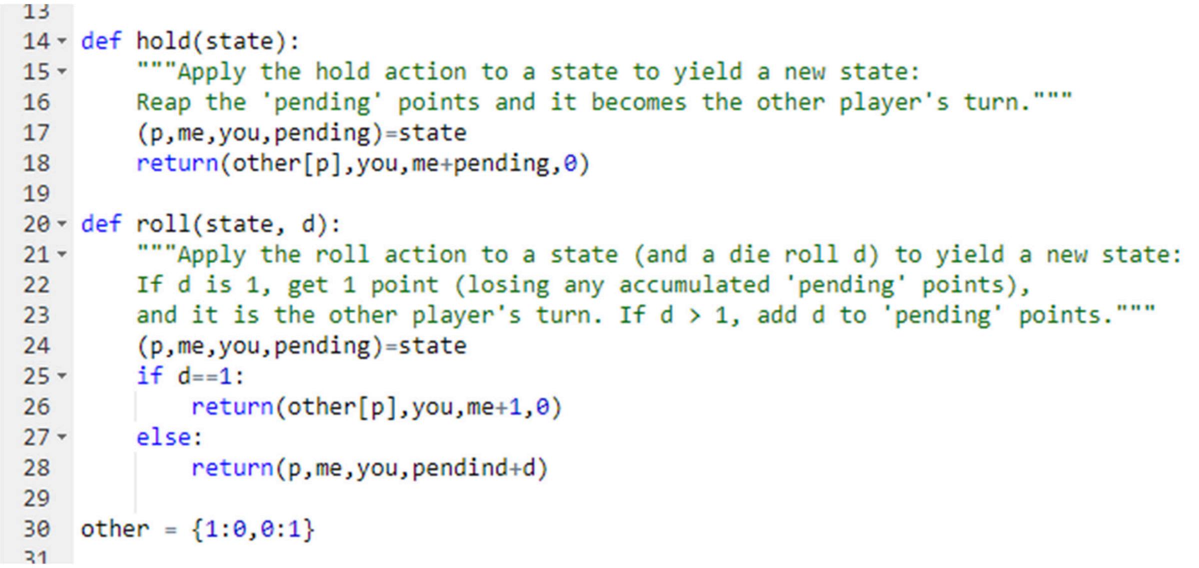
ACTIONS – roll / hold

LOW LEVEL

☑ Score 0
☑ Score 1
☑ pending
☑ player
□ previous roll
□ previous turn score

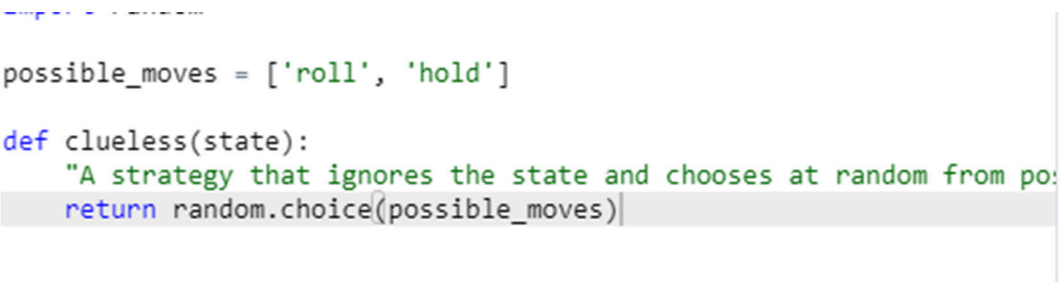
50 = goal

Hold And Roll



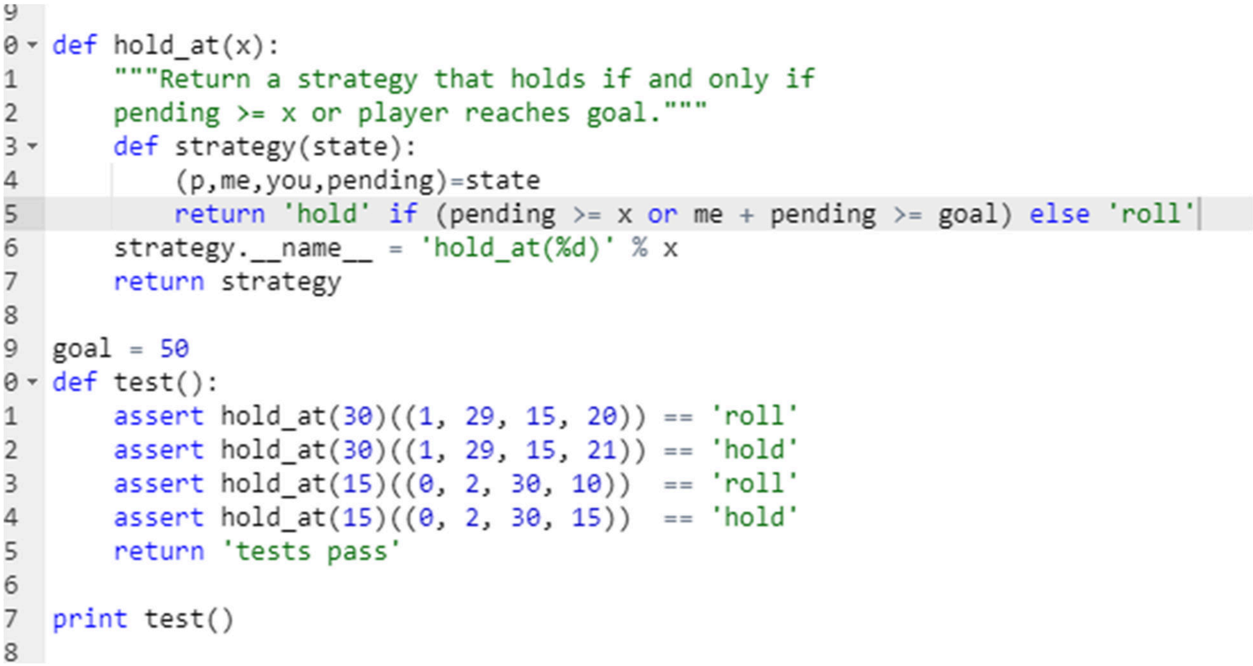
```
def hold(state):
    """Apply the hold action to a state to yield a new state:
    Reap the 'pending' points and it becomes the other player's turn."""
    (p,me,you,pending)=state
    return(other[p],you,me+pending,0)

def roll(state, d):
    """Apply the roll action to a state (and a die roll d) to yield a new state:
    If d is 1, get 1 point (losing any accumulated 'pending' points),
    and it is the other player's turn. If d > 1, add d to 'pending' points."""
    (p,me,you,pending)=state
    if d==1:
        return(other[p],you,me+1,0)
    else:
        return(p,me,you,pendind+d)

other = {1:0,0:1}
```

Clueless



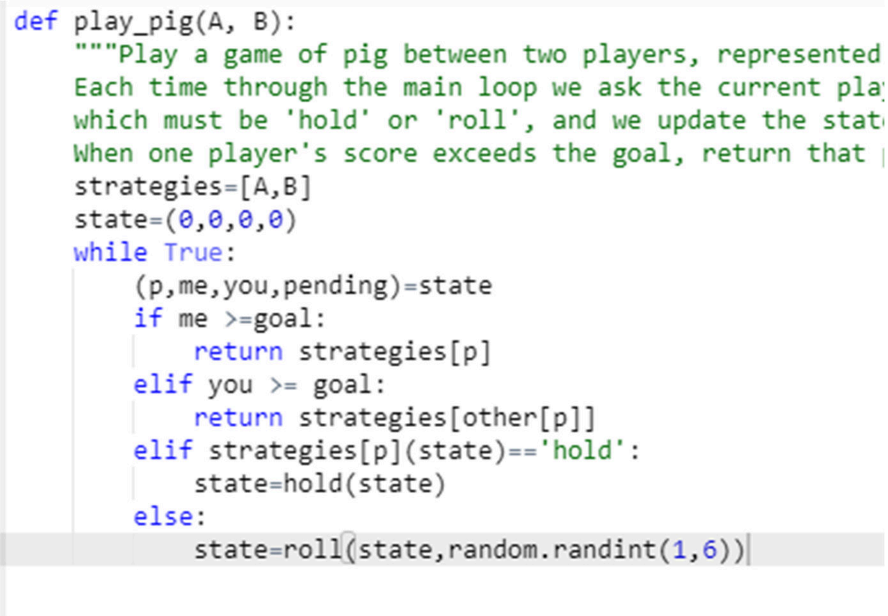
```
import random

possible_moves = ['roll', 'hold']

def clueless(state):
    "A strategy that ignores the state and chooses at random from po
    return random.choice(possible_moves)
```

Hold At Strategy



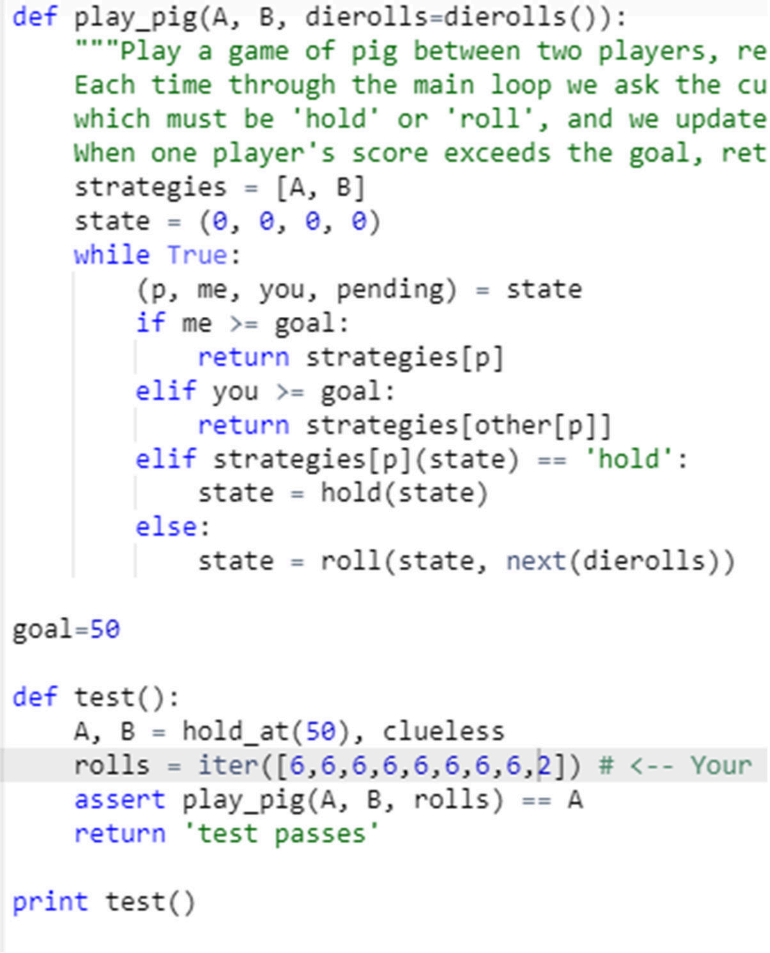
```
def hold_at(x):
    """Return a strategy that holds if and only if
    pending >= x or player reaches goal."""
    def strategy(state):
        (p,me,you,pending)=state
        return 'hold' if (pending >= x or me + pending >= goal) else 'roll'
    strategy.__name__ = 'hold_at(%d)' % x
    return strategy

goal = 50
def test():
    assert hold_at(30)((1, 29, 15, 20)) == 'roll'
    assert hold_at(30)((1, 29, 15, 21)) == 'hold'
    assert hold_at(15)((0, 2, 30, 10))  == 'roll'
    assert hold_at(15)((0, 2, 30, 15))  == 'hold'
    return 'tests pass'

print test()
```

Play Pig



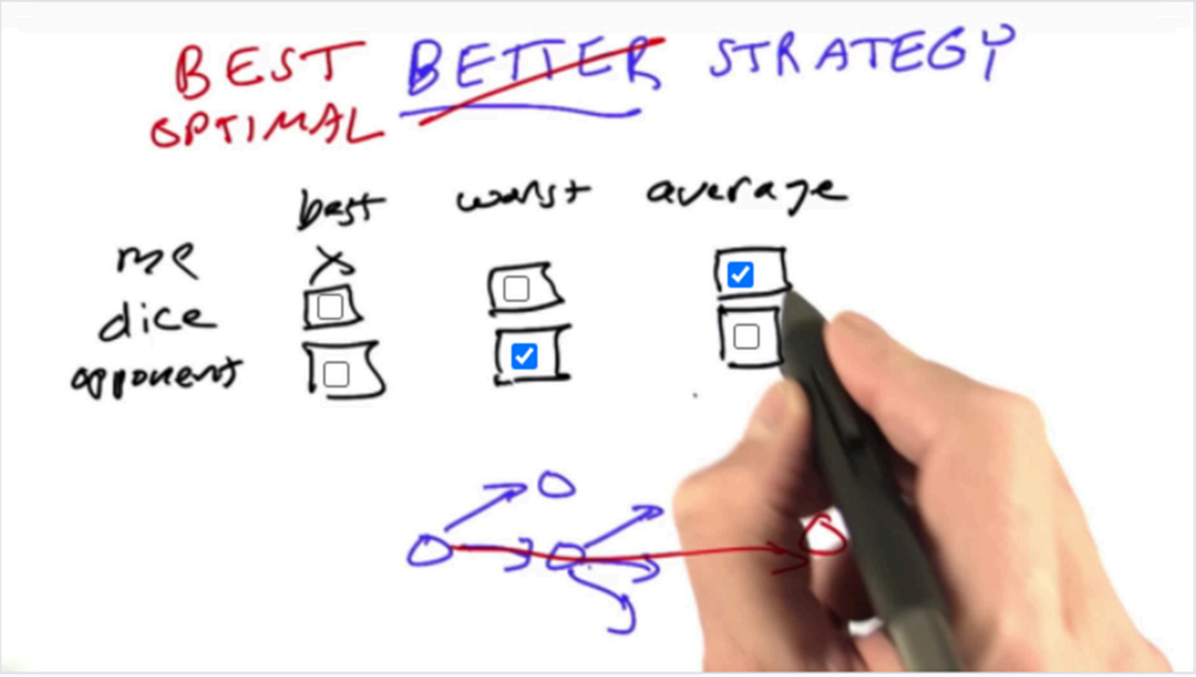
```
def play_pig(A, B):
    """Play a game of pig between two players, represented
    Each time through the main loop we ask the current pla
    which must be 'hold' or 'roll', and we update the stat
    When one player's score exceeds the goal, return that
    strategies=[A,B]
    state=(0,0,0,0)
    while True:
        (p,me,you,pending)=state
        if me >=goal:
            return strategies[p]
        elif you >= goal:
            return strategies[other[p]]
        elif strategies[p](state)=='hold':
            state=hold(state)
        else:
            state=roll(state,random.randint(1,6))
```

Loading The Dice



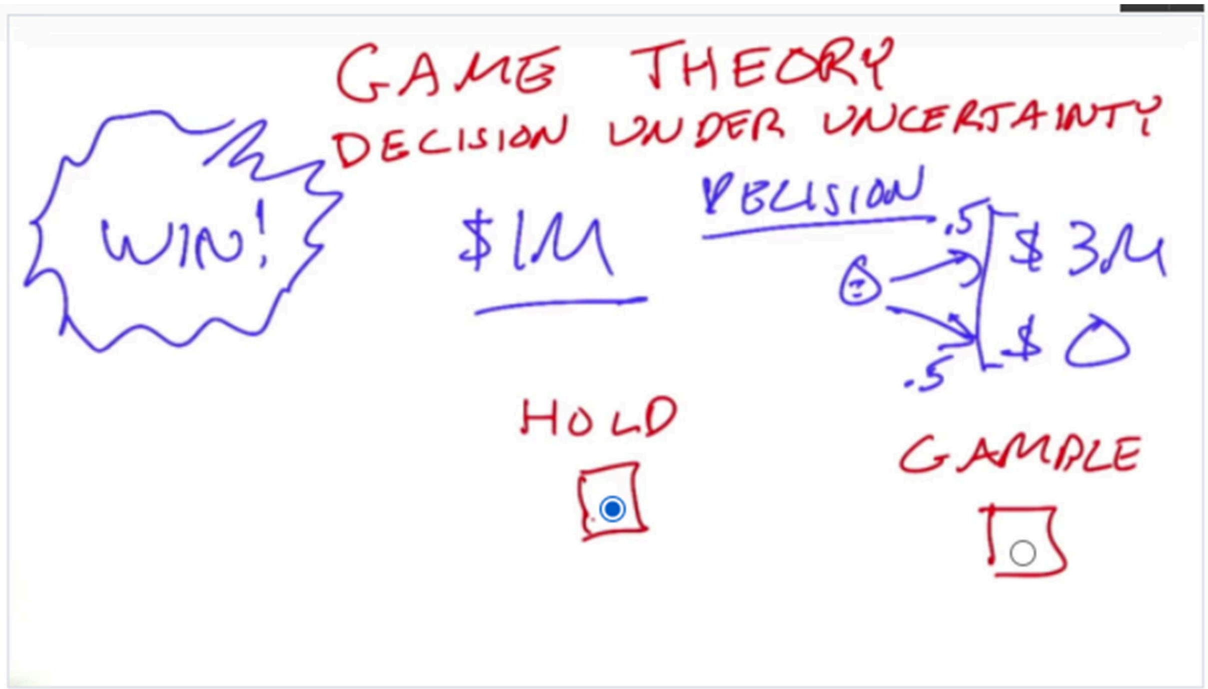
```
def play_pig(A, B, dierolls=dierolls()):
    """Play a game of pig between two players, re
    Each time through the main loop we ask the cu
    which must be 'hold' or 'roll', and we update
    When one player's score exceeds the goal, ret
    strategies = [A, B]
    state = (0, 0, 0, 0)
    while True:
        (p, me, you, pending) = state
        if me >= goal:
            return strategies[p]
        elif you >= goal:
            return strategies[other[p]]
        elif strategies[p](state) == 'hold':
            state = hold(state)
        else:
            state = roll(state, next(dierolls))

goal=50

def test():
    A, B = hold_at(50), clueless
    rolls = iter([6,6,6,6,6,6,6,6,2]) # <-- Your
    assert play_pig(A, B, rolls) == A
    return 'test passes'

print test()
```

Optimizing Strategy



	best	worst	average
me	X	☐	☑
dice	☐		☐
opponent	☐	☑	

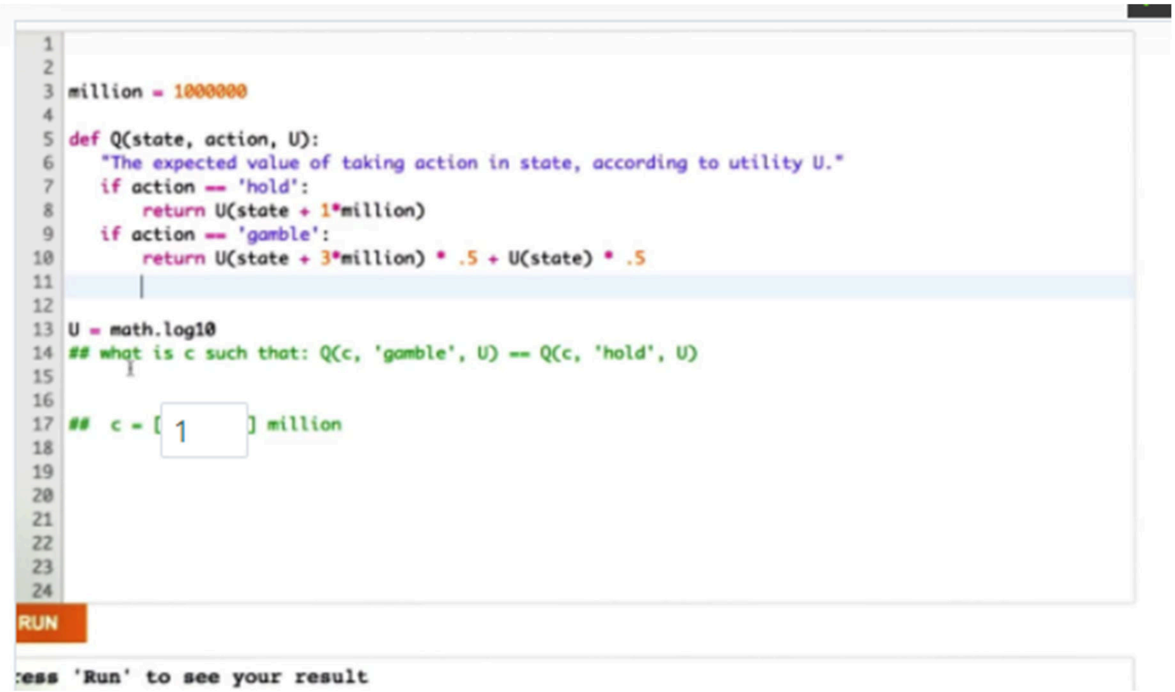
Game Theory



HOLD ●

GAMBLE ○

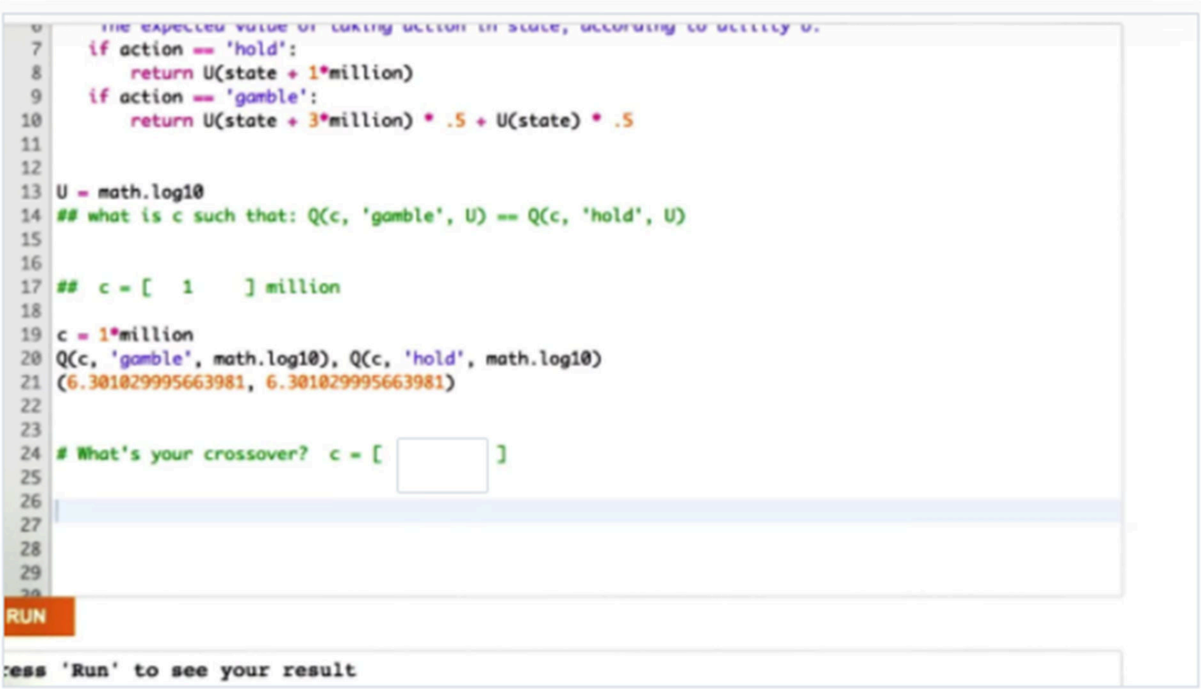
Break Even Point



```
million = 1000000

def Q(state, action, U):
    "The expected value of taking action in state, according to utility U."
    if action == 'hold':
        return U(state + 1*million)
    if action == 'gamble':
        return U(state + 3*million) * .5 + U(state) * .5


U = math.log10
## what is c such that: Q(c, 'gamble', U) == Q(c, 'hold', U)


##   c = [ 1      ] million
```

```
ess 'Run' to see your result
```

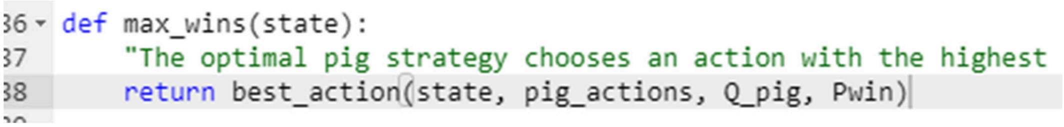
Whats Your Crossover



```
    "The expected value of taking action in state, according to utility U."
    if action == 'hold':
        return U(state + 1*million)
    if action == 'gamble':
        return U(state + 3*million) * .5 + U(state) * .5


U = math.log10
## what is c such that: Q(c, 'gamble', U) == Q(c, 'hold', U)


##   c = [   1     ] million

c = 1*million
Q(c, 'gamble', math.log10), Q(c, 'hold', math.log10)
(6.301029995663981, 6.301029995663981)


# What's your crossover?   c = [          ]
```

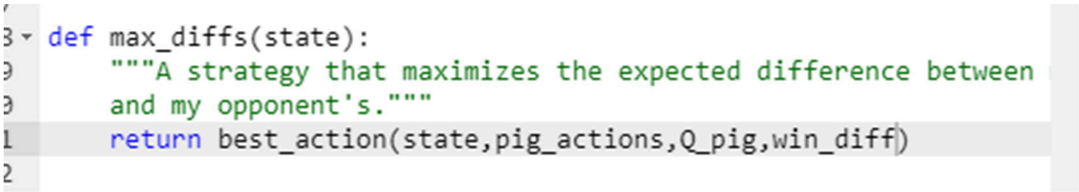
```
ess 'Run' to see your result
```

Maxwins



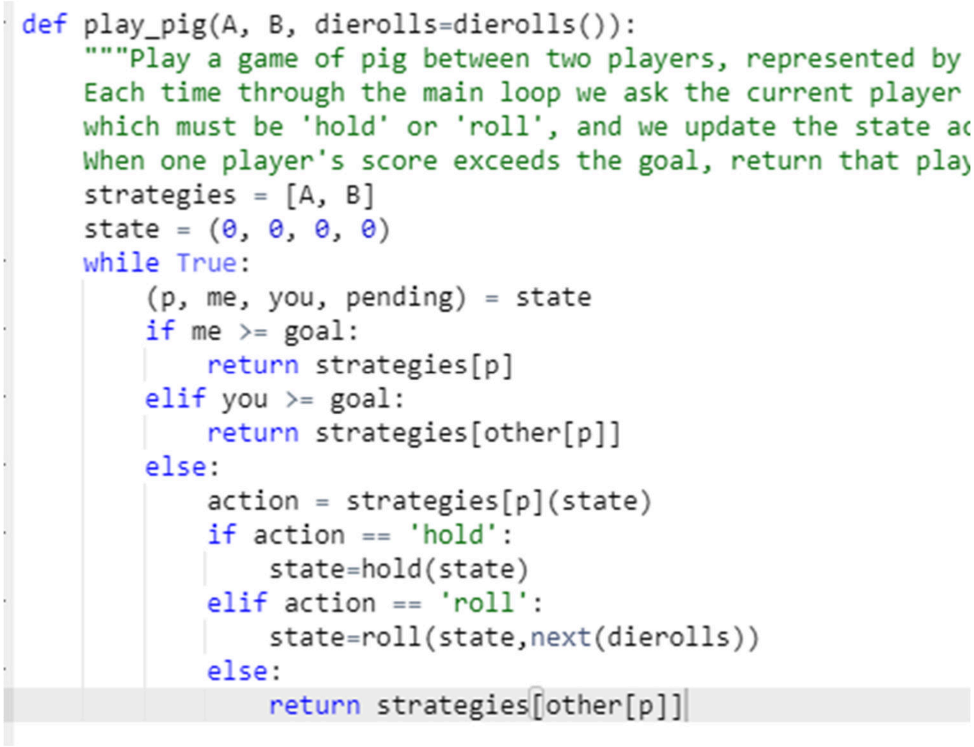
```
def max_wins(state):
    "The optimal pig strategy chooses an action with the highest
    return best_action(state, pig_actions, Q_pig, Pwin)
```

Maximizing Differential



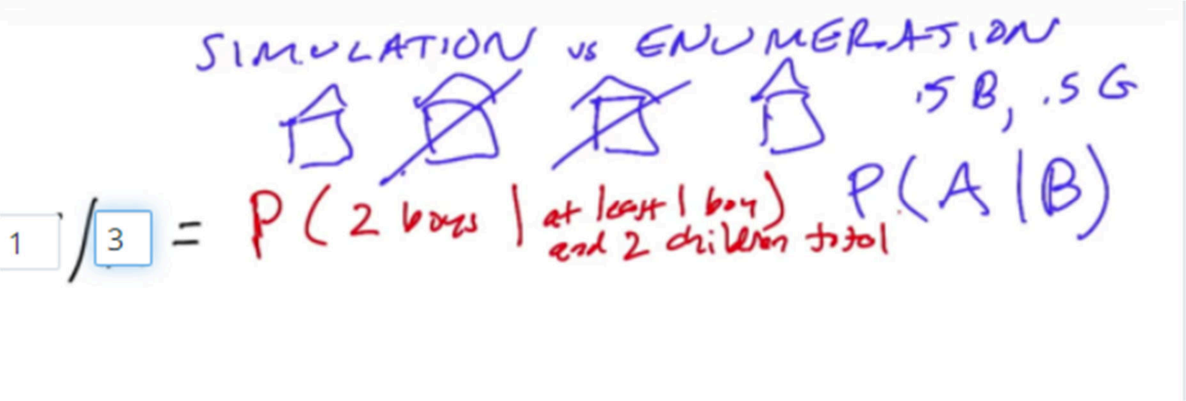
```
def max_diffs(state):
    """A strategy that maximizes the expected difference between
    and my opponent's."""
    return best_action(state,pig_actions,Q_pig,win_diff)
```

Legal Actions



```
def play_pig(A, B, dierolls=dierolls()):
    """Play a game of pig between two players, represented by
    Each time through the main loop we ask the current player
    which must be 'hold' or 'roll', and we update the state ac
    When one player's score exceeds the goal, return that play
    strategies = [A, B]
    state = (0, 0, 0, 0)
    while True:
        (p, me, you, pending) = state
        if me >= goal:
            return strategies[p]
        elif you >= goal:
            return strategies[other[p]]
        else:
            action = strategies[p](state)
            if action == 'hold':
                state=hold(state)
            elif action == 'roll':
                state=roll(state,next(dierolls))
            else:
                return strategies[other[p]]
```

Simulation Vs Enumeration

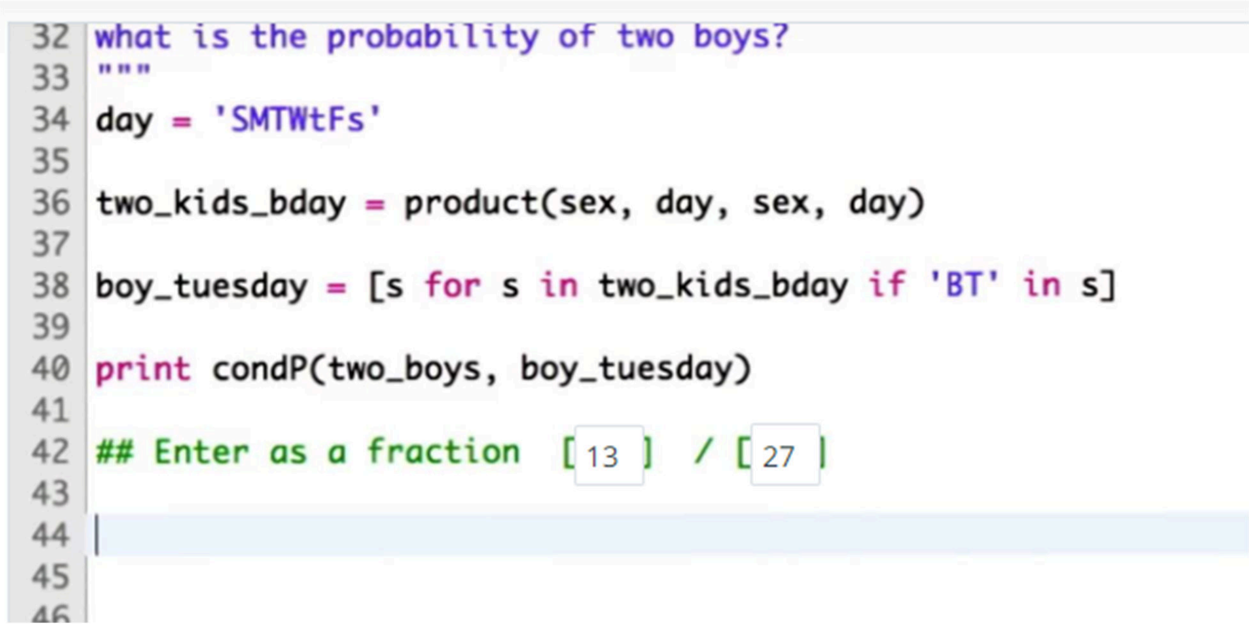


SIMULATION vs ENUMERATION

.5 B, .5 G

1 / 3 = P (2 boys | at least 1 boy and 2 children total) P(A | B)

Tuesday



```
what is the probability of two boys?
"""
day = 'SMTWtFs'

two_kids_bday = product(sex, day, sex, day)

boy_tuesday = [s for s in two_kids_bday if 'BT' in s]

print condP(two_boys, boy_tuesday)

## Enter as a fraction  [ 13 ]  / [ 27 ]


```