

Representing Hands

REPRESENTING HANDS

- ❑ `['JS', 'JD', '2S', '2C', '7H']`
- ❑ `[(11, 'S'), (11, 'D'), (2, 'S'), (2, 'C'), (7, 'H')]`
- ❑ `set(['JS', 'JD', '2S', '2C', '7H'])`
- ❑ `"JS JD 2S 2C 7H"`

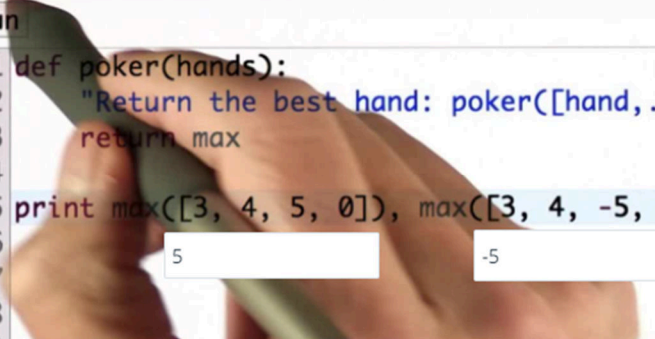
Poker Function

Run

```
1 def poker(hands):  
2     "Return the best hand: poker([hand,...]) => hand"  
3     return max  
4  
5  
6  
7  
8  
9
```

Understanding Max

```
Run
1 def poker(hands):
2     "Return the best hand: poker([hand,...]) => hand"
3     return max
4
5 print max([3, 4, 5, 0]), max([3, 4, -5, 0], key=abs)
6
7
8
9
```



The image shows a hand holding a green pen, pointing at the code. The code is displayed in a window with a light blue background. The pen is pointing at the line `print max([3, 4, 5, 0]), max([3, 4, -5, 0], key=abs)`. Below this line, there are two input boxes: the first contains the number 5, and the second contains the number -5.

Using Max

```
def poker(hands):
    "Return the best hand: poker([hand,...]) => hand"
    return max(hands, key=hand_rank)
```

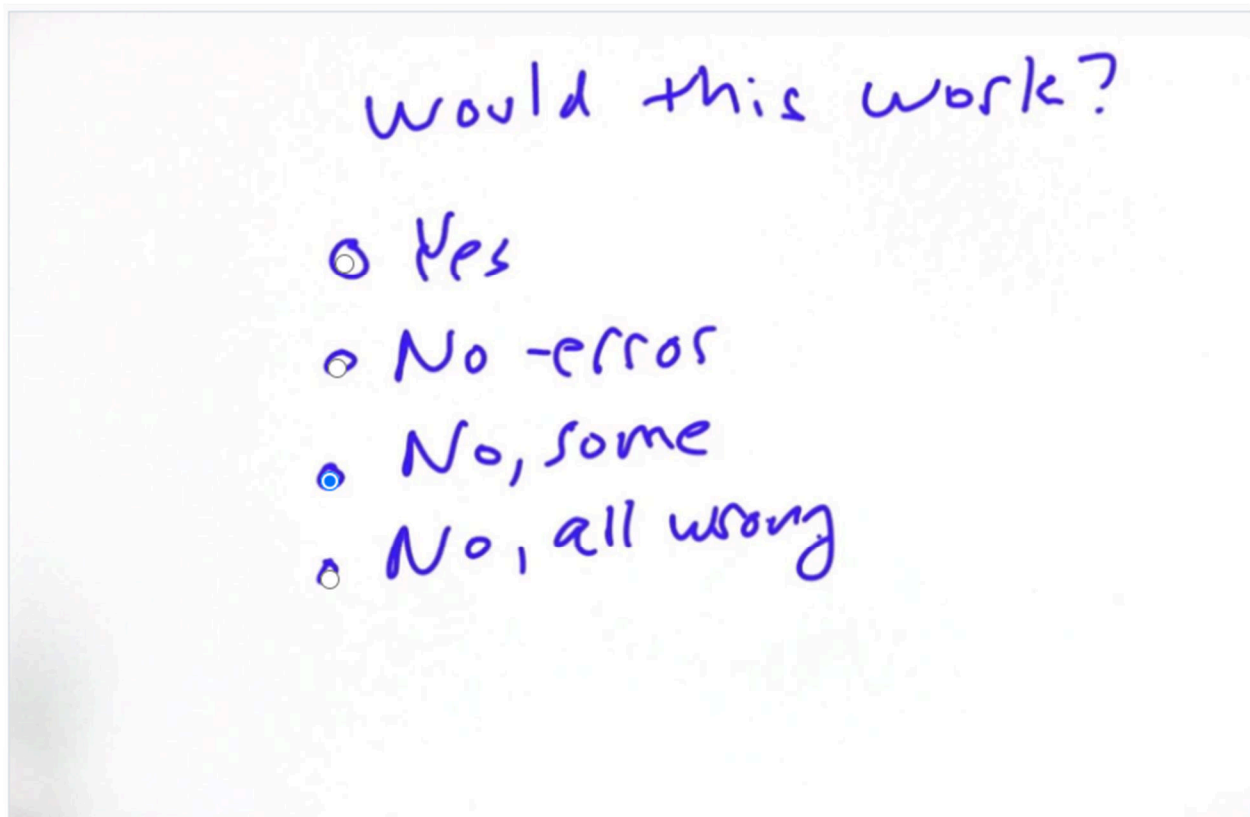
Testing

```
def test():
    "Test cases for the functions in poker program"
    sf = "6C 7C 8C 9C TC".split() # => ['6C', '7C', '8C', '9C', 'TC']
    fk = "9D 9H 9S 9C 7D".split()
    fh = "TD TC TH 7C 7D".split()
    assert poker([sf, fk, fh]) == sf
    assert poker([fk, fh]) == fk
    assert poker([fh, fh]) == fh
    return "test pass"
```

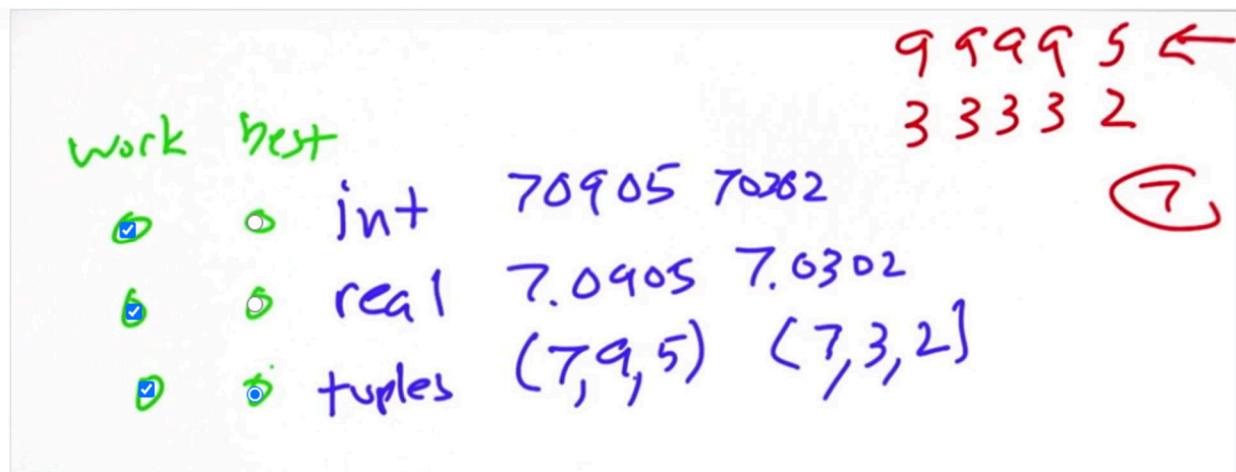
Extreme Values

```
def test():
    "Test cases for the functions in poker program"
    sf = "6C 7C 8C 9C TC".split()
    fk = "9D 9H 9S 9C 7D".split()
    fh = "TD TC TH 7C 7D".split()
    assert poker([sf, fk, fh]) == sf
    assert poker([fk, fh]) == fk
    assert poker([fh, fh]) == fh
    assert poker([sf]) == sf
    assert poker([sf]+99*[fh]) == sf
    return "test pass"
```

Hand Rank Attempt



Representing Rank



```
def test():
    "Test cases for the functions in poker program"
    sf = "6C 7C 8C 9C TC".split() # Straight Flush
    fk = "9D 9H 9S 9C 7D".split() # Four of a Kind
    fh = "TD TC TH 7C 7D".split() # Full House
    assert poker([sf, fk, fh]) == sf
    assert poker([fk, fh]) == fk
    assert poker([fh, fh]) == fh
    assert poker([sf]) == sf
    assert poker([sf] + 99*[fh]) == sf
    assert hand_rank(sf) == (8,10)
    assert hand_rank(fk) == (7,9,7)
    assert hand_rank(fh) == (6,10,7)
    print test()
```

```
> def hand_rank(hand):  
    ranks = card_ranks(hand)  
    > if straight(ranks) and flush(hand): # straight flush  
        return (8, max(ranks))  
    > elif kind(4, ranks): # 4 of a kind  
        return (7, kind(4, ranks), kind(1, ranks))  
    > elif kind(3, ranks) and kind(2, ranks): # full house  
        return (6, kind(3, ranks), kind(2, ranks))  
    > elif flush(hand): # flush  
        return (5, ranks)  
    > elif straight(ranks): # straight  
        return (4, max(ranks))  
    > elif kind(3, ranks): # 3 of a kind  
        return (3, kind(3, ranks), ranks)  
    > elif two_pair(ranks): # 2 pair  
        return (2, two_pair(ranks), ranks)  
    > elif kind(2, ranks): # kind  
        return (1, kind(2, ranks), ranks)  
    > else: # high card  
        return (0, ranks)
```

Testing Card Rank

```
5
6 def test():
7     "Test cases for the functions in poker program"
8     sf = "6C 7C 8C 9C TC".split() # Straight Flush
9     fk = "9D 9H 9S 9C 7D".split() # Four of a Kind
10    fh = "TD TC TH 7C 7D".split() # Full House
11    assert card_ranks(sf)==[10,9,8,7,6]
12    assert card_ranks(fk)==[9,9,9,9,7]
13    assert card_ranks(fh)==[10,10,10,7,7]
14    assert poker([sf, fk, fh]) == sf
15    assert poker([fk, fh]) == fk
16    assert poker([fh, fh]) == fh
17    assert poker([sf]) == sf
18    assert poker([sf] + 99*[fh]) == sf
19    assert hand_rank(sf) == (8, 10)
20    assert hand_rank(fk) == (7, 9, 7)
21    assert hand_rank(fh) == (6, 10, 7)
22    return 'tests pass'
23
```

Fixing Card Rank

```
def card_ranks(hand):
    "Return a list of the ranks, sorted with a higher first."
    ranks = ['--23456789TJQKA'.index(r) for r,s in hand]
    ranks.sort(reverse=True)
    return ranks

print card_ranks(['AC', '3D', '4S', 'KH']) #should output [14, 13, 4, 3]
```

Straight And Flush

```
8 def straight(ranks):
9     "Return True if the ordered ranks form a 5-card straight."
10    return (max(ranks)-min(ranks) == 4) and len(set(ranks))==5
11
12 def flush(hand):
13     "Return True if all the cards have the same suit."
14     suits = [s for r,s in hand]
15     return len(set(suits))== 1
16
```

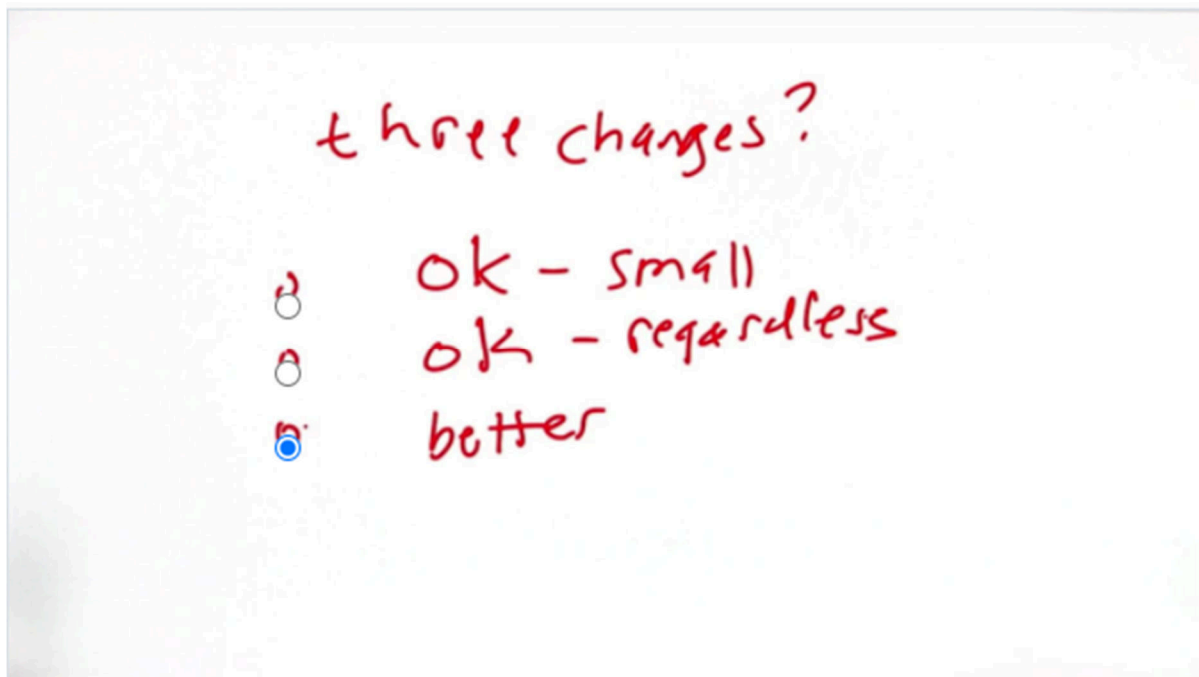
Kind function

```
6 def kind(n, ranks):
7     """Return the first rank that this hand has exactly n of.
8     Return None if there is no n-of-a-kind in the hand."""
9     for r in ranks:
10         if ranks.count(r) == n: return r
11     return None
```

Two Pair Function

```
5
6 def two_pair(ranks):
7     """If there are two pair, return the two ranks as a
8     tuple: (highest, lowest); otherwise return None."""
9     pair=kind(2,ranks)
10    lowpair=kind(2,list(reversed(ranks)))
11    if pair and lowpair != pair:
12        return (pair, lowpair)
13    else:
14        return None
```

Making Changes



What to Change

change what?

- o poker
- o hand-rank
- o card-ranks
- o straight

Ace Low Straight

```
def card_ranks(hand):  
    "Return a list of the ranks, sorted with higher first."  
    ranks = ['--23456789TJQKA'.index(r) for r, s in hand]  
    ranks.sort(reverse = True)  
    return [5,4,3,2,1] if (ranks==[14,5,4,3,2]) else ranks
```

Handling Ties

handle ties?

- ok next
- o hand-rank
 - o poker
 - o new function

Allmax

```
1 def allmax(iterable, key=None):
2     "Return a list of all items equal to the max of the iterable."
3     result, maxval = [], None
4     key = key or (lambda x: x)
5     for x in iterable:
6         xval = key(x)
7         if not result or xval > maxval:
8             result, maxval = [x], xval
9         elif xval == maxval:
10            result.append(x)
11    return result
```

Deal

```
1 def deal(numhands, n=5, deck=mydeck):
2     random.shuffle(deck)
3     return [deck[n*i:n*(i+1)] for i in range(numhands)]
```

Hand Frequencies

