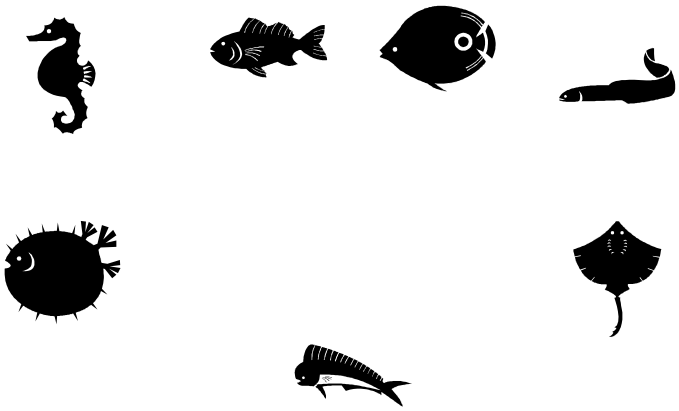
Sets and Dictionaries

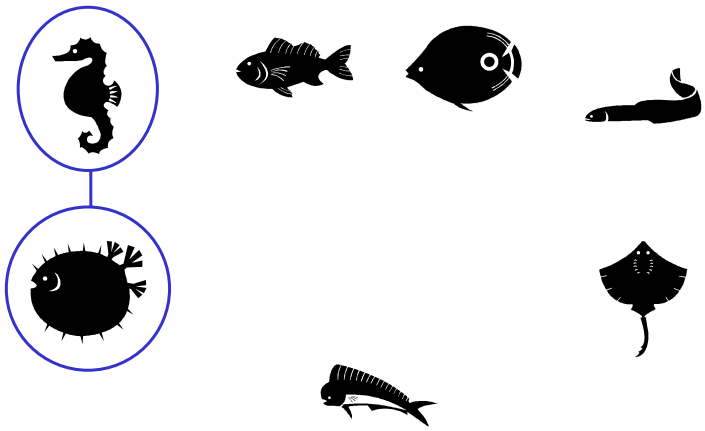
Phylogenetic Trees

Some organisms are more alike than others

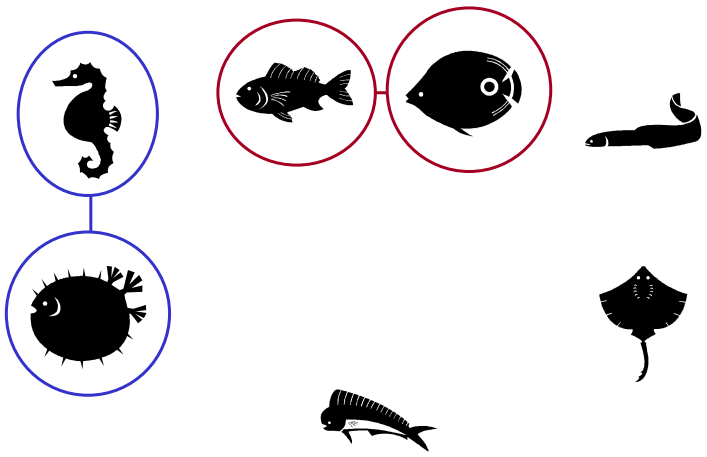
Some organisms are more alike than others



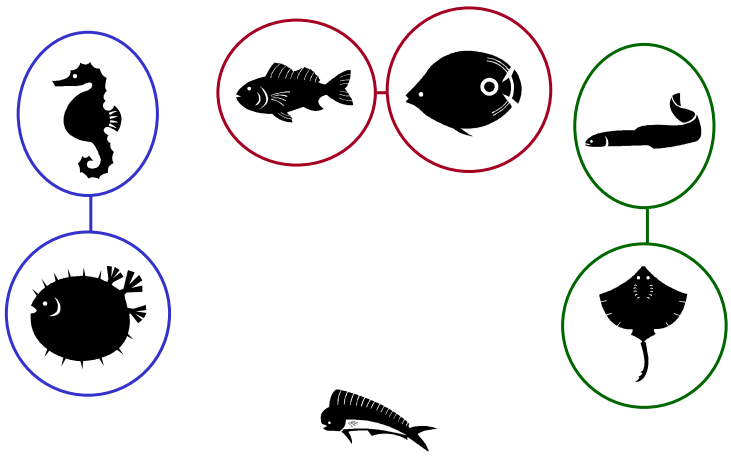
Some organisms are more alike than others



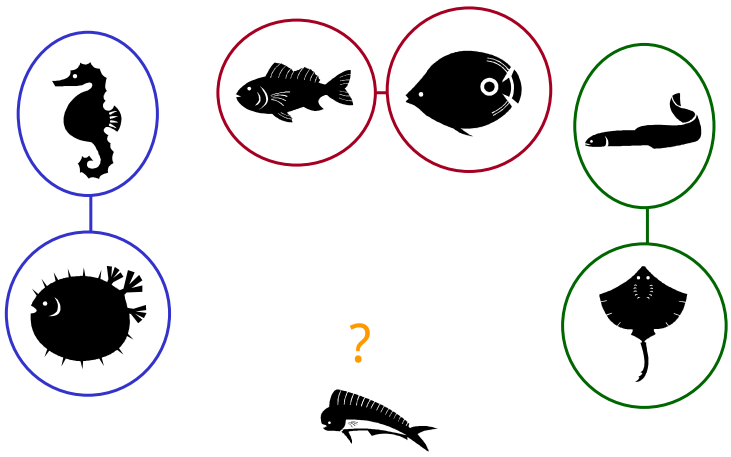
Some organisms are more alike than others



Some organisms are more alike than others



Some organisms are more alike than others



Nothing in biology makes sense except in the light of evolution.

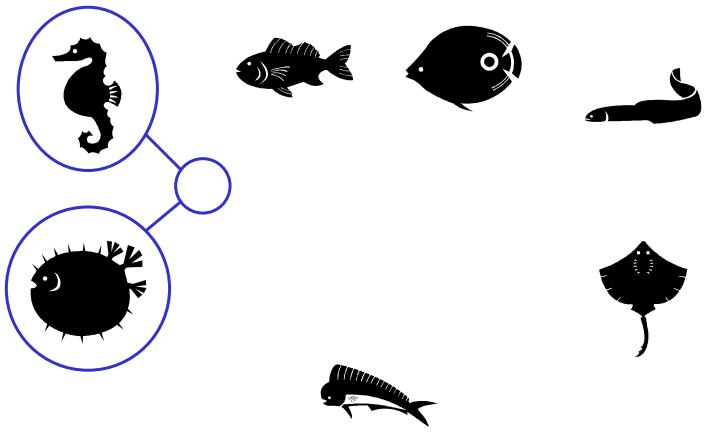
— Theodosius Dobzhansky

The closer their DNA, the more recently they had a common ancestor

Reconstruct their evolutionary tree using a *hierarchical clustering algorithm*

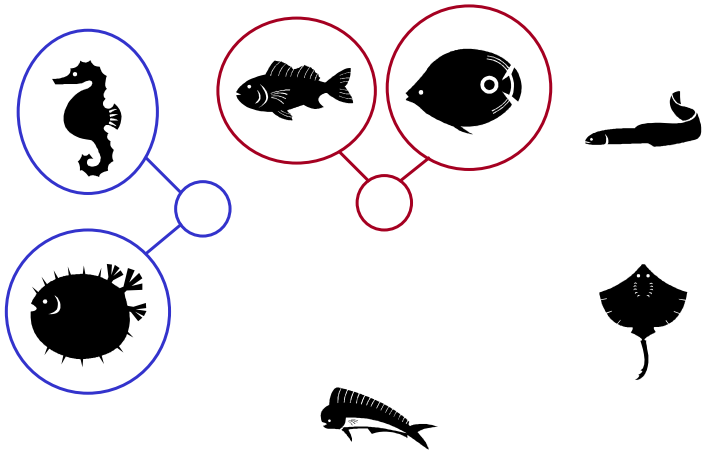
software carpentry

Calculate the common ancestor of the two closest

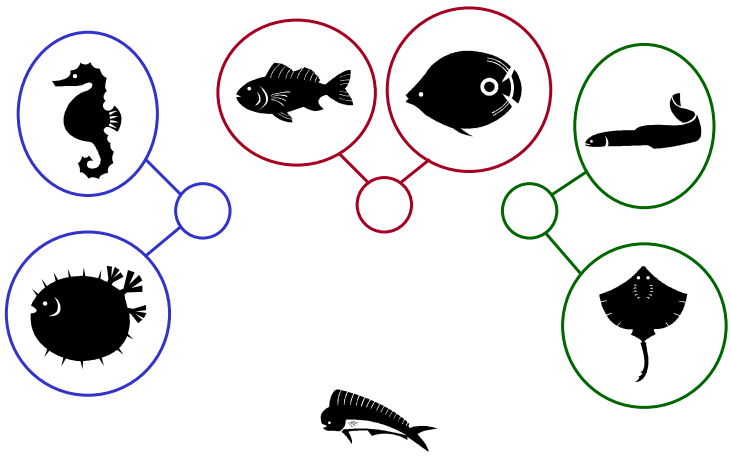


software carpentry

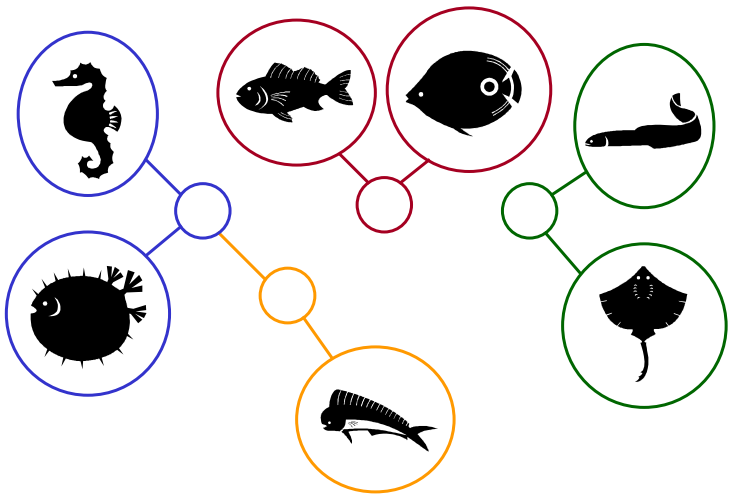
Then of the next closest



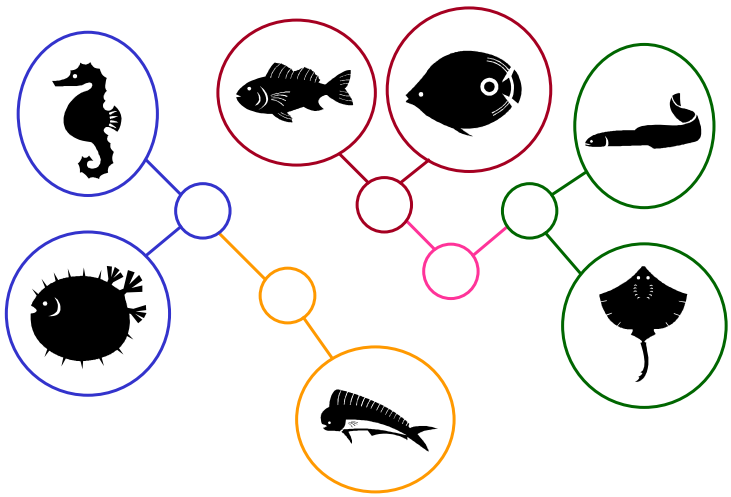
And the next



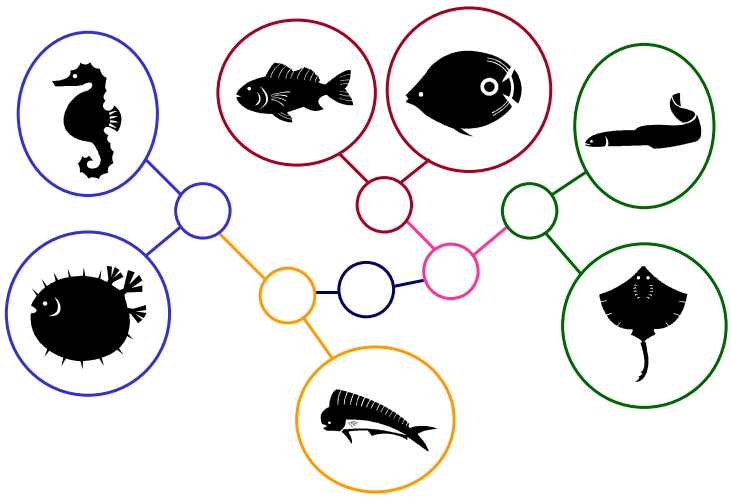
Combine organisms with common ancestors



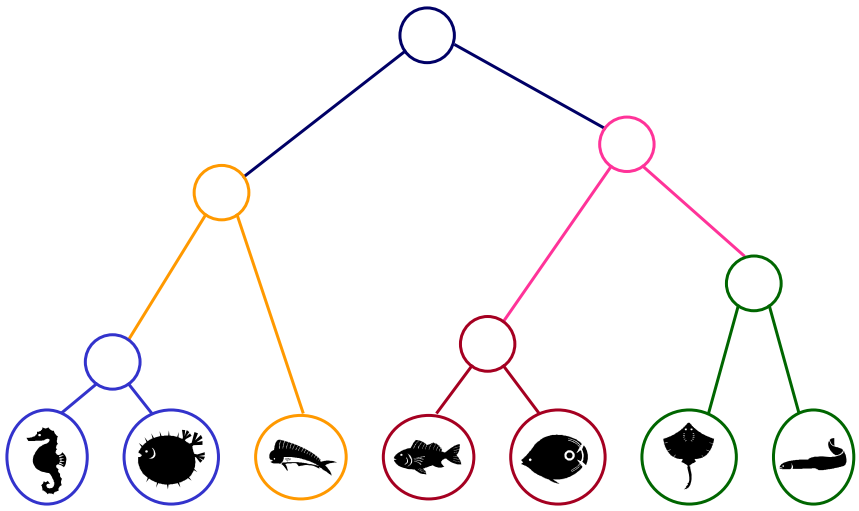
And common ancestors with each other



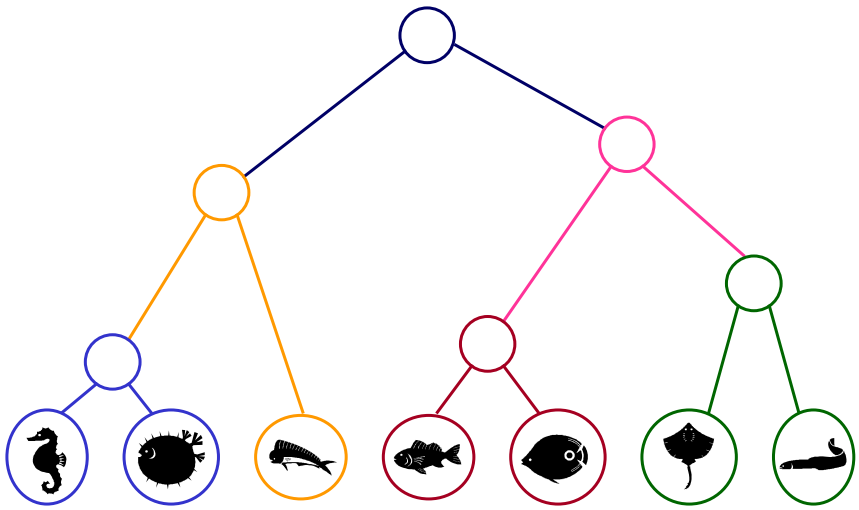
To construct a complete tree



Redraw



Redraw using height to show number of differences



Turn this into an algorithm

Turn this into an algorithm

```
U = {all organisms}
while U ≠ {}:
  a, b = two closest entries in U
  p = common parent of {a, b}
  U = U - {a, b}
  U = U + {p}
```

Turn this into an algorithm

```
U = {all organisms}
while U ≠ {}:
  a, b = two closest entries in U
  p = common parent of {a, b}
  U = U - {a, b}
  U = U + {p}
```

Ungrouped set shrinks by one element each time

Turn this into an algorithm

```
U = {all organisms}
while U ≠ {}:
  a, b = two closest entries in U
  p = common parent of {a, b}
  U = U - {a, b}
  U = U + {p}
```

Ungrouped set shrinks by one element each time

Keep track of pairings on the side to draw tree later

What does "closest" mean?

What does "closest" mean?

Simplest algorithm is *unweighted pair-group method using arithmetic averages* (UPGMA)

What does "closest" mean?

Simplest algorithm is *unweighted pair-group method using arithmetic averages* (UPGMA)

	human	vampire	werewolf	mermaid
human				
vampire	13			
werewolf	5	6		
mermaid	12	15	29	

Closest entries are human (H) and werewolf (W)

	H	V	W	M
H				
V	13			
W	5	6		
M	12	15	29	

Closest entries are human (H) and werewolf (W)

Replace with HW (common ancestor)

	H	V	W	M
H				
V	13			
W	5	6		
M	12	15	29	

Closest entries are human (H) and werewolf (W)

Replace with HW (common ancestor)

Height is 1/2 value of entry

	H	V	W	M
H				
V	13			
W	5	6		
M	12	15	29	

Closest entries are human (H) and werewolf (W)

Replace with HW (common ancestor)

Height is 1/2 value of entry

Replace score for X with (HX + WX - HW)/2

	H	V	W	M
H				
V	13			
W	5	6		
M	12	15	29	

Closest entries are human (H) and werewolf (W)

Replace with HW (common ancestor)

Height is 1/2 value of entry

Replace score for X with (HX + WX - HW)/2

	H	V	W	M
H				
V	13			
W	5	6		
M	12	15	29	

Closest entries are human (H) and werewolf (W)

Replace with HW (common ancestor)

Height is 1/2 value of entry

Replace score for X with (HX + WX - HW)/2

	H	V	W	M
H				
V	13			
W	5	6		
M	12	15	29	

	HW	V	M
HW			
V	7		
M	18	15	

Closest entries are human (H) and werewolf (W)

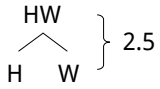
Replace with HW (common ancestor)

Height is 1/2 value of entry

Replace score for X with (HX + WX - HW)/2

	H	V	W	M
H				
V	13			
W	5	6		
M	12	15	29	

	HW	V	M
HW			
V	7		
M	18	15	

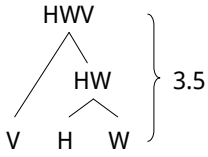


Sets and Dictionaries Phylogenetic Trees

Repeat

	HW	V	M
HW			
V	7		
M	18	15	

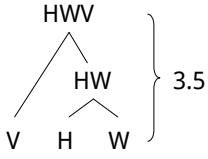
	HWV	M
HWV		
M	13	



Repeat

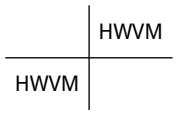
	HW	V	M
HW			
V	7		
M	18	15	

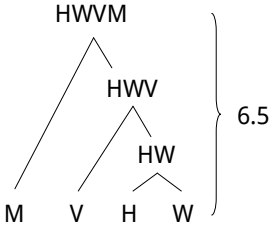
	HWV	M
HWV		
M	13	



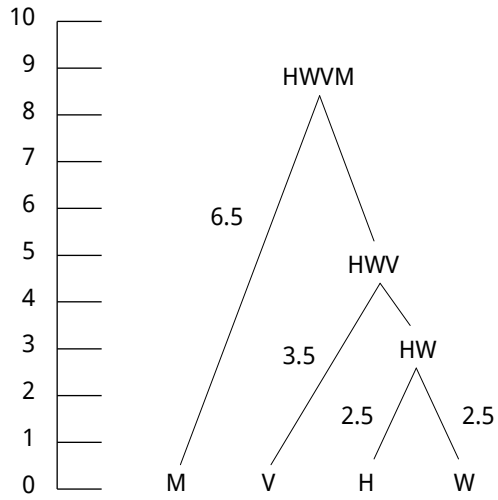
And again

	HWV	M
HWV		
M	13	

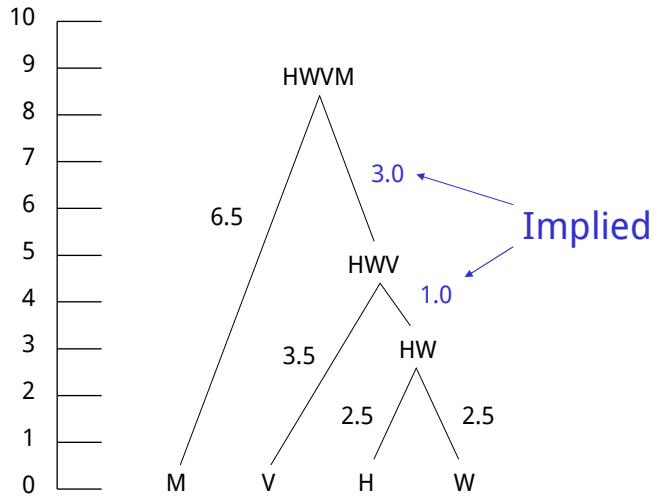




Final tree



Final tree



How to translate this into software?

How to translate this into software?

We drew it as a triangular matrix...

How to translate this into software?

We drew it as a triangular matrix...

...but the order of the rows and columns is arbitrary

How to translate this into software?

We drew it as a triangular matrix...

...but the order of the rows and columns is arbitrary

It's really just a lookup table...

How to translate this into software?

We drew it as a triangular matrix...

...but the order of the rows and columns is arbitrary

It's really just a lookup table...

...so we should think about using a dictionary

How to translate this into software?

We drew it as a triangular matrix...

...but the order of the rows and columns is arbitrary

It's really just a lookup table...

...so we should think about using a dictionary

Key: (organism, organism)

How to translate this into software?

We drew it as a triangular matrix...

...but the order of the rows and columns is arbitrary

It's really just a lookup table...

...so we should think about using a dictionary

Key: (organism, organism)

- In alphabetical order to ensure uniqueness

How to translate this into software?

We drew it as a triangular matrix...

...but the order of the rows and columns is arbitrary

It's really just a lookup table...

...so we should think about using a dictionary

Key: (organism, organism)

- In alphabetical order to ensure uniqueness

Value: distance

	H	V	W	M
H				
V	13			
W	5	6		
M	12	15	29	

	H	V	W	M
H				
V	13			
W	5	6		
M	12	15	29	

```
{
    ('human',    'mermaid')  : 12,
    ('human',    'vampire')  : 13,
    ('human',    'werewolf') :  5,
    ('mermaid', 'vampire')  : 15,
    ('mermaid', 'werewolf') : 29,
    ('vampire', 'werewolf') :  6
}
```

Write out the algorithm

Write out the algorithm

```
while len(scores) > 0:
  min_pair = find_min_pair(species, scores)
  parent, height = create_new_parent(scores, min_pair)
  print parent, height
  old_score = remove_entries(species, scores, min_pair)
  update(species, scores, min_pair, parent, old_score)
```

Write out the algorithm

```
while len(scores) > 0:
  min_pair = find_min_pair(species, scores)
  parent, height = create_new_parent(scores, min_pair)
  print parent, height
  old_score = remove_entries(species, scores, min_pair)
  update(species, scores, min_pair, parent, old_score)
```

Assumes scores are in a dictionary `scores`

Write out the algorithm

```
while len(scores) > 0:
  min_pair = find_min_pair(species, scores)
  parent, height = create_new_parent(scores, min_pair)
  print parent, height
  old_score = remove_entries(species, scores, min_pair)
  update(species, scores, min_pair, parent, old_score)
```

Assumes scores are in a dictionary `scores`

And species names are in a list `species`

Write out the algorithm

```
while len(scores) > 0:
  min_pair = find_min_pair(species, scores)
  parent, height = create_new_parent(scores, min_pair)
  print parent, height
  old_score = remove_entries(species, scores, min_pair)
  update(species, scores, min_pair, parent, old_score)
```

Assumes scores are in a dictionary `scores`

And species names are in a list `species`

And yes, we revised this a couple of times...


```
def find_min_pair(species, scores):
  '''Find minimum-value pair of species in scores.'''

  min_pair, min_val = None, None
  for pair in combos(species):
    assert pair in scores, \
           'Pair (%s, %s) not in scores' % pair
    if (min_val is None) or (scores[pair] < min_val):
      min_pair, min_val = pair, scores[pair]

  assert min_val is not None, \
         'No minimum value found in scores'
  return min_pair
```

```
def find_min_pair(species, scores):
  '''Find minimum-value pair of species in scores.'''

  min_pair, min_val = None, None
  for pair in combos(species):
    assert pair in scores, \
           'Pair (%s, %s) not in scores' % pair
    if (min_val is None) or (scores[pair] < min_val):
      min_pair, min_val = pair, scores[pair]

  assert min_val is not None, \
         'No minimum value found in scores'
  return min_pair
```



```
def find_min_pair(species, scores):
  '''Find minimum-value pair of species in scores.'''

  min_pair, min_val = None, None
  for pair in combos(species):
    assert pair in scores, \
           'Pair (%s, %s) not in scores' % pair
    if (min_val is None) or (scores[pair] < min_val):
      min_pair, min_val = pair, scores[pair]

  assert min_val is not None, \
         'No minimum value found in scores'
  return min_pair
```

```
def combos(species):
  '''Generate all combinations of species.'''

  result = []
  for i in range(len(species)):
    for j in range(i+1, len(species)):
      result.append((species[i], species[j]))

  return result
```



```
def combos(species):
  '''Generate all combinations of species.'''

  result = []
  for i in range(len(species)):
    for j in range(i+1, len(species)):
      result.append((species[i], species[j]))

  return result
```

Sets and Dictionaries Phylogenetic Trees

```
def combos(species):
  '''Generate all combinations of species.'''

  result = []
  for i in range(len(species)):
    for j in range(i+1, len(species)):
      result.append((species[i], species[j]))

  return result
```

```
def combos():
def find_min_pair():
if __name__ == '__main__':
  ...main program...
```



```
def create_new_parent(scores, pair):
  '''Create record for new parent.'''

  parent = '[%s %s]' % pair
  height = scores[pair] / 2.
  return parent, height
```

```
def create_new_parent(scores, pair):
  '''Create record for new parent.'''

  parent = '[%s %s]' % pair
  height = scores[pair] / 2.
  return parent, height
```



```
def create_new_parent(scores, pair):
  '''Create record for new parent.'''

  parent = '[%s %s]' % pair
  height = scores[pair] / 2.
  return parent, height
```


```
def combos():
def find_min_pair():
def create_new_parent():
if __name__ == '__main__':
  ...main program...
```

```
def remove_entries(species, scores, pair):
  '''Remove species that have been combined.'''

  left, right = pair
  species.remove(left)
  species.remove(right)
  old_score = scores[pair]
  del scores[pair]
  return old_score
```



```
def remove_entries(species, scores, pair):
  '''Remove species that have been combined.'''

  left, right = pair
  species.remove(left)
  species.remove(right)
  old_score = scores[pair]
  del scores[pair]
  return old_score
```

Sets and Dictionaries Phylogenetic Trees

```
def remove_entries(species, scores, pair):
  '''Remove species that have been combined.'''

  left, right = pair
  species.remove(left)
  species.remove(right)
  old_score = scores[pair]
  del scores[pair]
  return old_score
```

```
def combos():
def find_min_pair():
def create_new_parent():
def remove_entries():
if __name__ == '__main__':
  ...main program...
```



```
def update(species, scores, pair, parent, parent_score):
  '''Replace two species from the scores table.'''

  left, right = pair
  for other in species:
    l_score = tidy_up(scores, left, other)
    r_score = tidy_up(scores, right, other)
    new_pair = make_pair(parent, other)
    new_score = (l_score + r_score - parent_score)/2.
    scores[new_pair] = new_score

  species.append(parent)
  species.sort()
```

```
def update(species, scores, pair, parent, parent_score):
  '''Replace two species from the scores table.'''

  left, right = pair
  for other in species:
    l_score = tidy_up(scores, left, other)
    r_score = tidy_up(scores, right, other)
    new_pair = make_pair(parent, other)
    new_score = (l_score + r_score - parent_score)/2.
    scores[new_pair] = new_score

  species.append(parent)
  species.sort()
```



```
def update(species, scores, pair, parent, parent_score):
  '''Replace two species from the scores table.'''

  left, right = pair
  for other in species:
    l_score = tidy_up(scores, left, other)
    r_score = tidy_up(scores, right, other)
    new_pair = make_pair(parent, other)
    new_score = (l_score + r_score - parent_score)/2.
    scores[new_pair] = new_score

  species.append(parent)
  species.sort()
```

Sets and Dictionaries Phylogenetic Trees

```
def update(species, scores, pair, parent, parent_score):
  '''Replace two species from the scores table.'''

  left, right = pair
  for other in species:
    l_score = tidy_up(scores, left, other)
    r_score = tidy_up(scores, right, other)
    new_pair = make_pair(parent, other)
    new_score = (l_score + r_score - parent_score)/2.
    scores[new_pair] = new_score

  species.append(parent)
  species.sort()
```



```
def update(species, scores, pair, parent, parent_score):
  '''Replace two species from the scores table.'''

  left, right = pair
  for other in species:
    l_score = tidy_up(scores, left, other)
    r_score = tidy_up(scores, right, other)
    new_pair = make_pair(parent, other)
    new_score = (l_score + r_score - parent_score)/2.
    scores[new_pair] = new_score

  species.append(parent)
  species.sort()
```

Sets and Dictionaries Phylogenetic Trees

```
def update(species, scores, pair, parent, parent_score):
  '''Replace two species from the scores table.'''

  left, right = pair
  for other in species:
    l_score = tidy_up(scores, left, other)
    r_score = tidy_up(scores, right, other)
    new_pair = make_pair(parent, other)
    new_score = (l_score + r_score - parent_score)/2.
    scores[new_pair] = new_score

  species.append(parent)
  species.sort()
```



```
def update(species, scores, pair, parent, parent_score):
  '''Replace two species from the scores table.'''

  left, right = pair
  for other in species:
    l_score = tidy_up(scores, left, other)
    r_score = tidy_up(scores, right, other)
    new_pair = make_pair(parent, other)
    new_score = (l_score + r_score - parent_score)/2.
    scores[new_pair] = new_score

  species.append(parent)
  species.sort()
```

Sets and Dictionaries Phylogenetic Trees

```
def update(species, scores, pair, parent, parent_score):
  '''Replace two species from the scores table.'''

  left, right = pair
  for other in species:
    l_score = tidy_up(scores, left, other)
    r_score = tidy_up(scores, right, other)
    new_pair = make_pair(parent, other)
    new_score = (l_score + r_score - parent_score)/2.
    scores[new_pair] = new_score

  species.append(parent)
  species.sort()
```

```
def combos():
def find_min_pair():
def create_new_parent():
def remove_entries():
def update():
if __name__ == '__main__':
  ...main program...
```



```
def tidy_up(scores, old, other):
  '''Clean out references to old species.'''
  pair = make_pair(old, other)
  score = scores[pair]
  del scores[pair]
  return score
```

```
def tidy_up(scores, old, other):
  '''Clean out references to old species.'''
  pair = make_pair(old, other)
  score = scores[pair]
  del scores[pair]
  return score
```



```
def tidy_up(scores, old, other):
  '''Clean out references to old species.'''
  pair = make_pair(old, other)
  score = scores[pair]
  del scores[pair]
  return score

def make_pair(left, right):
  '''Make an ordered pair of species.'''

  if left < right: return (left, right)
  else:            return (right, left)
```

```
def tidy_up(scores, old, other):
  '''Clean out references to old species.'''
  pair = make_pair(old, other)
  score = scores[pair]
  del scores[pair]
  return score

def make_pair(left, right):
  '''Make an ordered pair of species.'''

  if left < right: return (left, right)
  else:            return (right, left)
```

```
def combos():
def find_min_pair():
def create_new_parent():
def remove_entries():
def make_pair():
def tidy_up():
def update():
if __name__ == '__main__':
  ...main program...
```



```
$ python phylogen.py
[human werewolf] 2.5
[[human werewolf] vampire] 3.5
[[[human werewolf] vampire] mermaid] 6.5
```

Exercise 1: write unit tests

Exercise 2: reconstruct entire tree

Exercise 3: why does `update` sort?

created by

Elango Cheran

November 2010