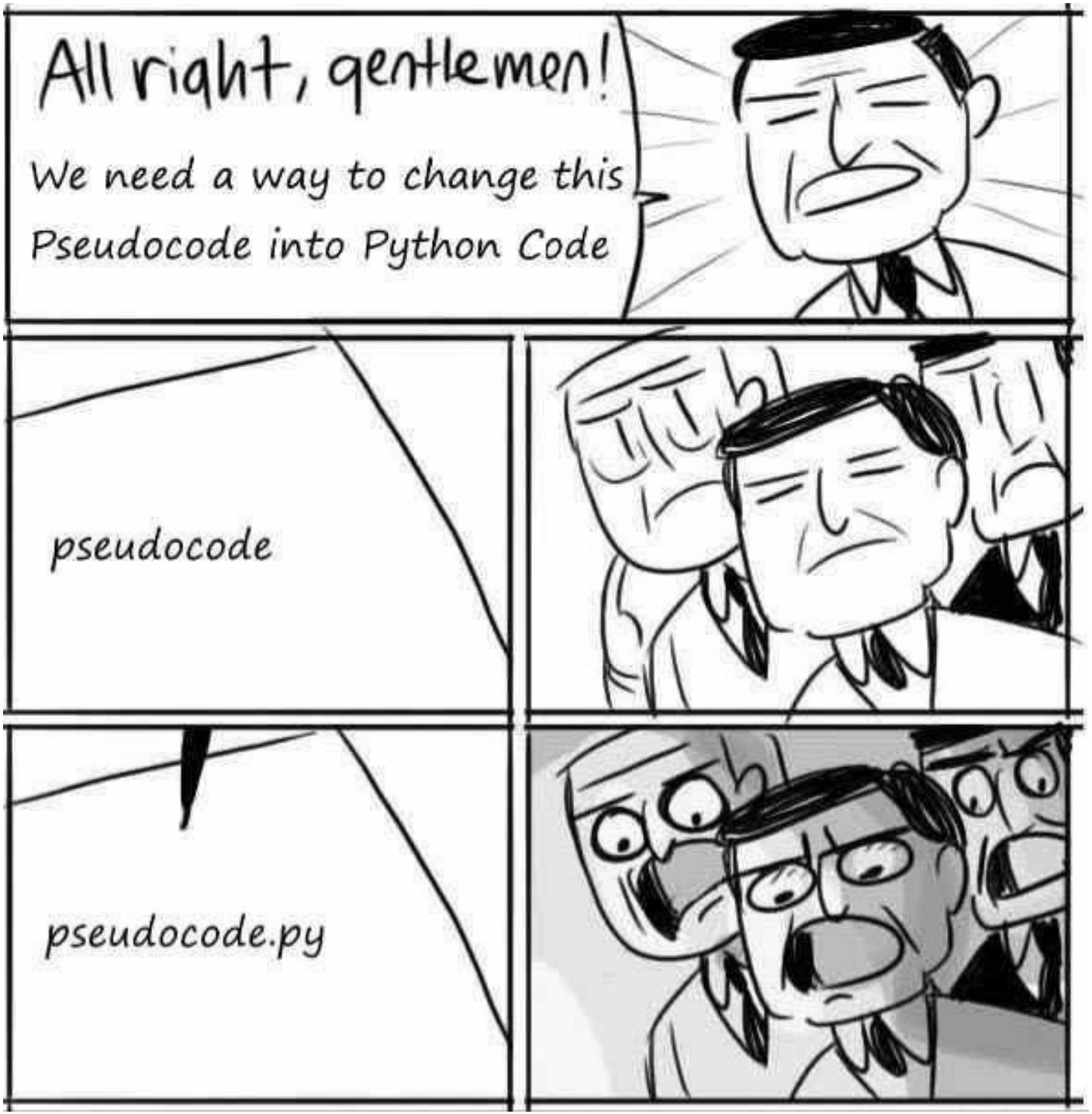

All right, gentlemen!
We need a way to change this Pseudocode into Python Code
pseudocode
pseudocode.py

Lessons from HW3: docstrings

Good docstring!

```
def fastLCS(stringA, stringB, memoD):
    """Accepts two strings and a dictionary as inputs. Returns
    the length of the Longest Common Substring as an int."""
```

Most people had them. Don’t forget them!

Remember to include info about the **inputs** and **outputs**!

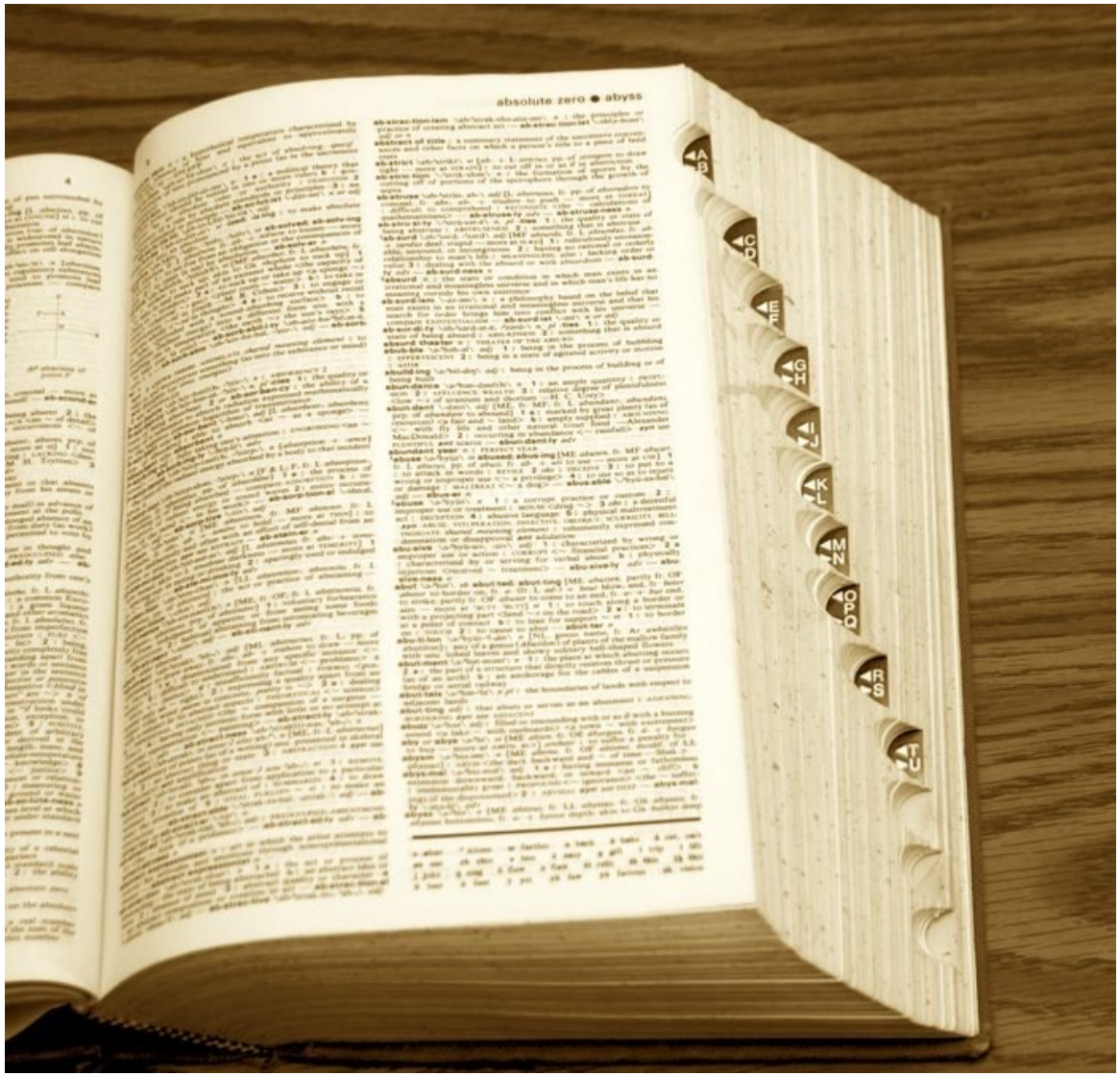
Useless docstring!

```
def fastLCS(stringA, stringB, memoD):
    """Calculates LCS, but fast."""
```

Lessons from HW3:

START EARLIER!!!

`dict` keys cannot be mutable



the *key*

Each word in a dictionary has a definition

the *value*

Words are stored in a way that enables easy look-up.

tuples are immutable lists

```
>>> L = [6,7,8,9]
>>> L
[6, 7, 8, 9]
>>> L[0]
6
>>> L[1:]
[7, 8, 9]
>>> L.append(42)
>>>
>>> L
[6, 7, 8, 9, 42]
>>> L + [43, "spam"]
[6, 7, 8, 9, 42, 43, "spam"]
>>> L + ["spam"]
[6, 7, 8, 9, 42, "spam"]
```

```
>>> T = (6,7,8,9)
>>> T
(6, 7, 8, 9)
>>> T[0]
6
>>> T[1:]
(7, 8, 9)
>>> T.append(42)
ERROR!
>>> T
(6, 7, 8, 9)
>>> T + (43, "spam")
(6, 7, 8, 9, 43, "spam")
>>> T + ("spam")
TypeError: can only concatenate
tuple (not "str") to tuple
>>> T + ("spam",)
(6, 7, 8, 9, "spam")
```

change revisted

```
def change(target, coinsL):
    '''Accepts an integer and a list as inputs. Returns the fewest
    number of coins needed to make the target integer.'''
    # base case 1: no coins required
    if target == 0: return 0

    # base case 2: impossible to make change
    elif coinsL == []: return float('inf')
    else:
        # discard coin if it exceeds the target
        if coinsL[0] > target:
            return change(target, coinsL[1:])

        # try both using and losing a coin; return minimum req'd
        else:
            useIt  = 1 + change(target - coinsL[0], coinsL)
            loseIt = change(target, coinsL[1:])
            return min(useIt, loseIt)
```

change revisted

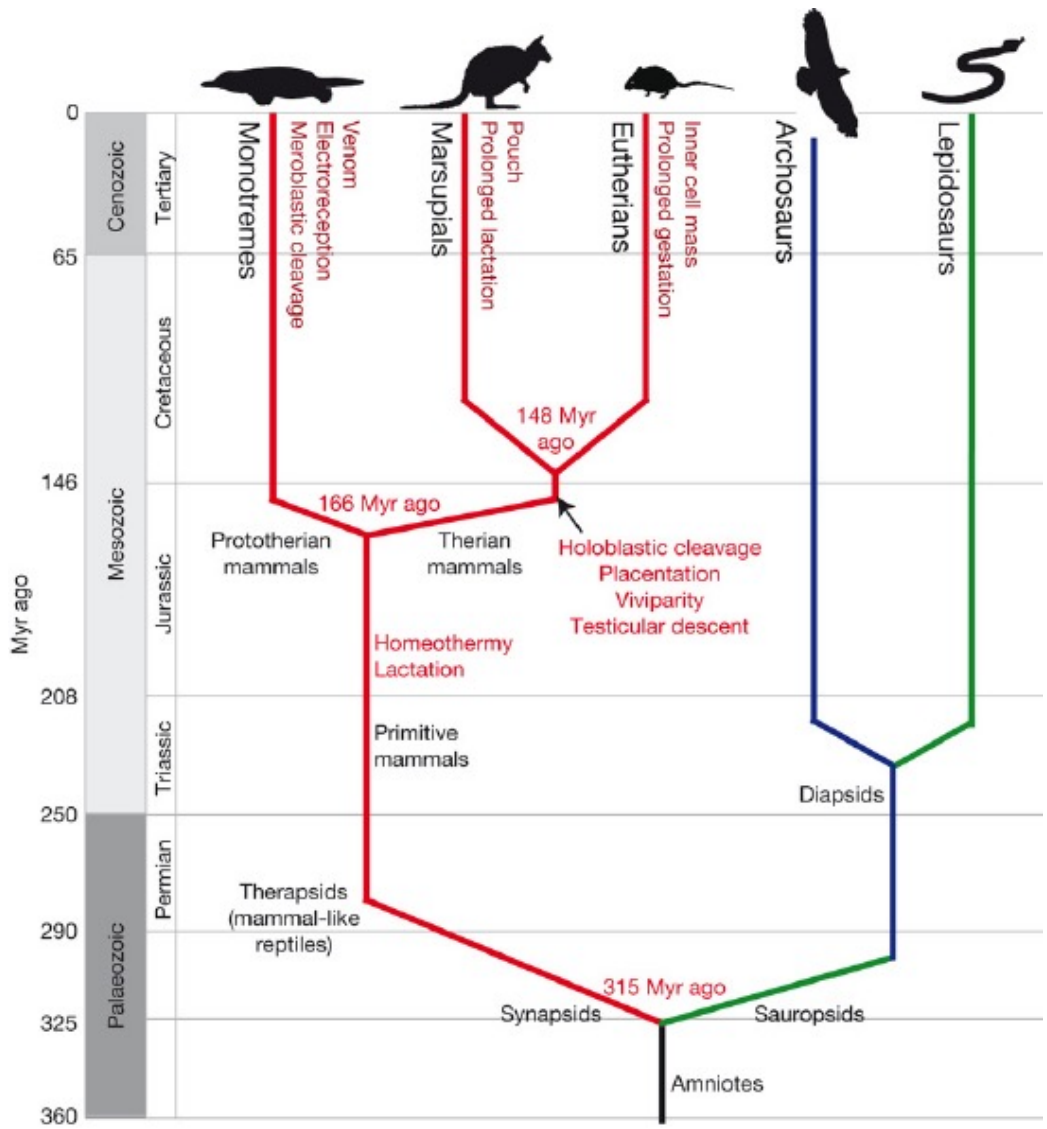
```
def change(target, coinsT):
    '''Accepts an integer and a tuple as inputs. Returns the fewest
    number of coins needed to make the target integer.'''
    # base case 1: no coins required
    if target == 0: return 0

    # base case 2: impossible to make change
    elif coinsT == (): return float('inf')
    else:
        # discard coin if it exceeds the target
        if coinsT[0] > target:
            return change(target, coinsT[1:])

        # try both using and losing a coin; return minimum req'd
        else:
            useIt  = 1 + change(target - coinsT[0], coinsT)
            loseIt = change(target, coinsT[1:])
            return min(useIt, loseIt)
```

Memoize change

Q

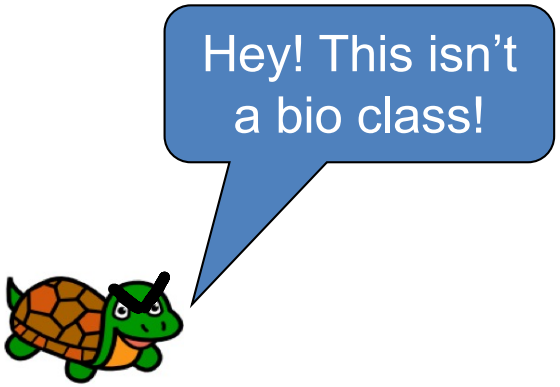
```
def memoChange(target, coinsT, memoD):
    '''Accepts an integer, a tuple, and a dictionary as inputs.
    Returns the fewest number of coins needed for the target.'''
```

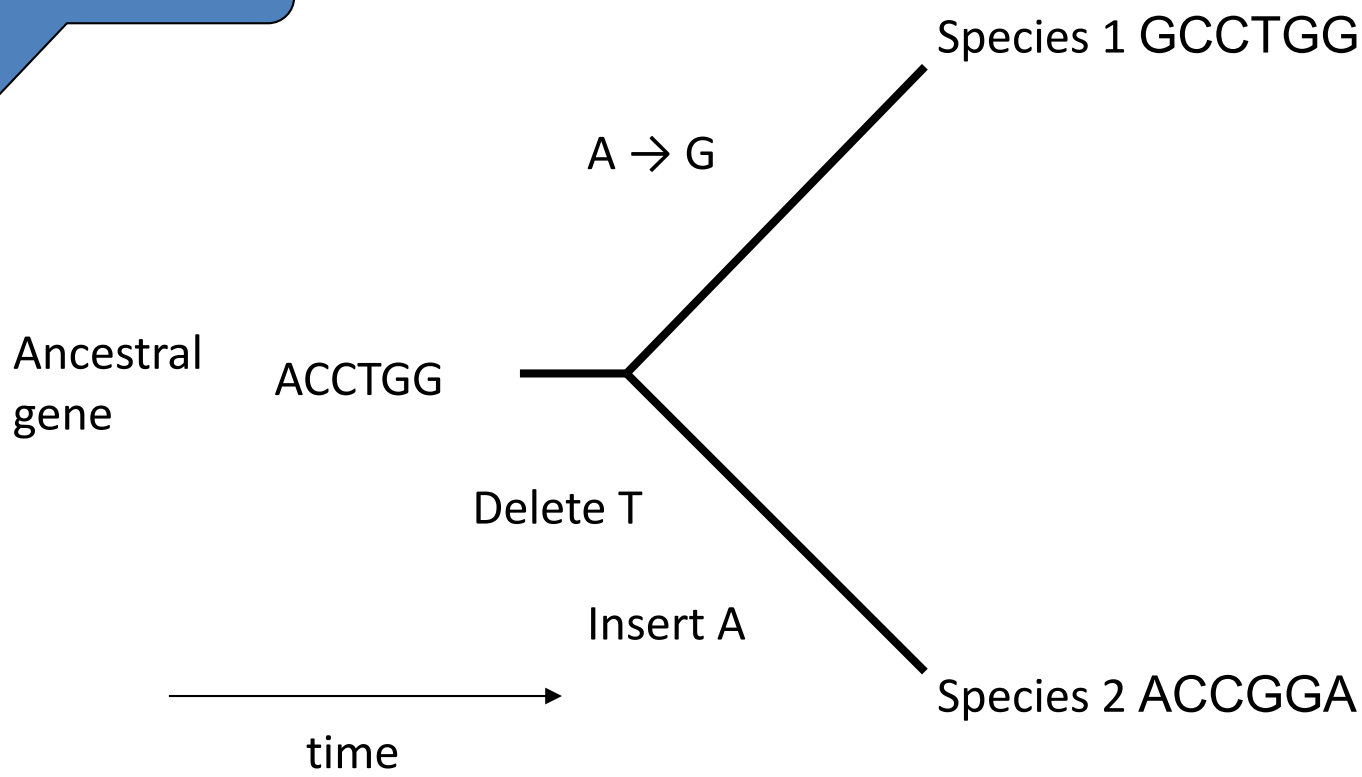
Is the mammalian X homologous to the avian Z?



Nature 453, 175-183 fig1

Homology in sequences





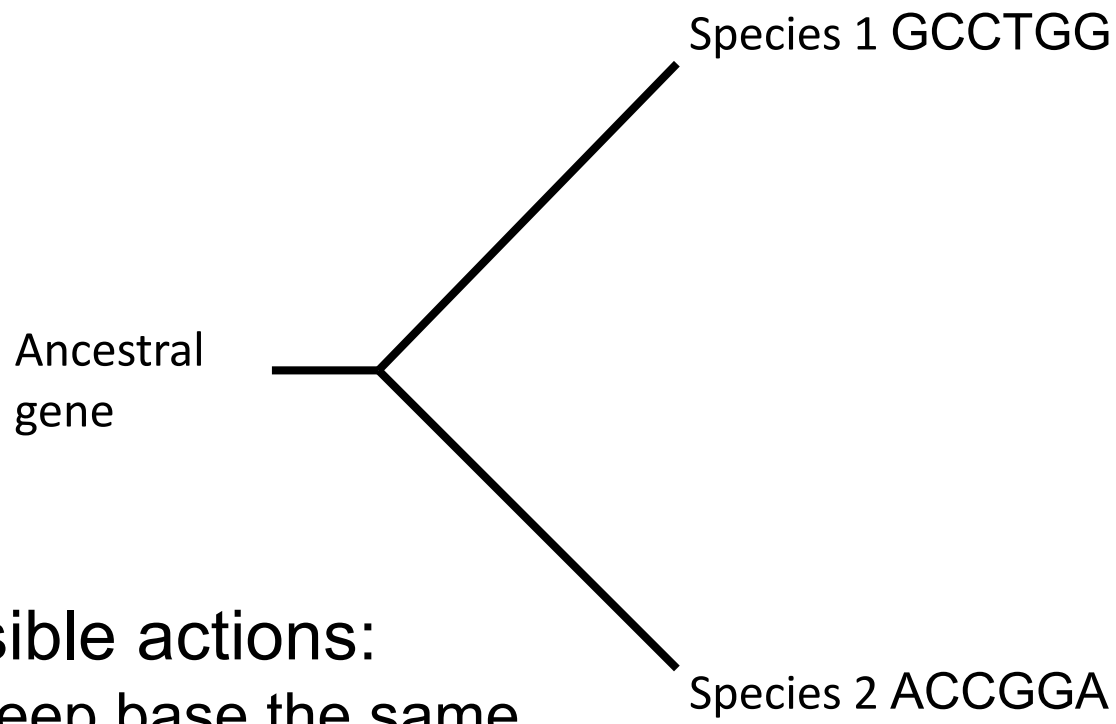
Orthologs: homologous genes with a shared evolutionary lineage

LCS ignores differences

```
>>> LCS('DAQS','MDAQ')
3


>>> LCS('DAQ','SDASQMMMMMMMMM')
3
```

This score doesn’t reflect the biological implications of changing one string of bases/amino acids to another.

A biologically informed similarity score



Possible actions:

- Keep base the same
- Insert new base
- Delete existing base
- Change one base to another

GCCTGG ➔ ACCGGA

GCCTGG → **A**CCTGG	(change G to A)
A**C**CTGG	(keep the C)
AC**C**TGG	(keep the C)
ACC**T**GG → ACCGG	(delete the T)
ACC**G**G	(keep the G)
ACCG**G**	(keep the G)
ACCGG → ACCGG**A**	(insert A)

GCCTGG → CCTGG	(delete the G)
CCTGG → CTGG	(delete the C)
CTGG →TGG	(delete the C)
TGG → GG	(delete the T)
GG → G	(delete the G)
G → ""	(delete the G)
"" → **A**	(insert an A)
A → A**C**	(insert a C)
AC → AC**C**	(insert a C)
ACC → ACC**G**	(insert a G)
ACCG → ACCG**G**	(insert a G)
ACCGG → ACCGG**A**	(insert an A)

>8900 ways to convert one 6bp sequence into another 6bp sequence. We want the best way.

Scoring editing operations

GCCTGG → **A**CCTGG	(change G to A)	**-1**
A**C**CTGG	(keep the C)	**+1**
AC**C**TGG	(keep the C)	**+1**
ACC**T**GG → ACCGG	(delete the T)	**-1**
ACC**G**G	(keep the G)	**+1**
ACCG**G**	(keep the G)	**+1**
ACCGG → ACCGG**A**	(insert A)	**-1**

score = 1

GCCTGG → CCTGG	(delete the G)	**-1**
CCTGG → CTGG	(delete the C)	**-1**
CTGG →TGG	(delete the C)	**-1**
TGG → GG	(delete the T)	**-1**
GG → G	(delete the G)	**-1**
G → ""	(delete the G)	**-1**
"" → **A**	(insert an A)	**-1**
A → A**C**	(insert a C)	**-1**
AC → AC**C**	(insert a C)	**-1**
ACC → ACC**G**	(insert a G)	**-1**
ACCG → ACCG**G**	(insert a G)	**-1**
ACCGG → ACCGG**A**	(insert an A)	**-1**

score = -12

Actions get rewards (+) and costs (-):

- Keep base the same: +1
- Insert new base: -1
- Delete existing base: -1
- Change one base to another: -1

alignScore

```
>>> alignScore("GCCTGG","ACCGGA")
1

>>> alignScore("spam","scramble")
-2
```

```
S1: GCCTGG-
S2: ACC-GGA
```

These are called “alignments”

```
S1: sp-am---
S2: scramble
```

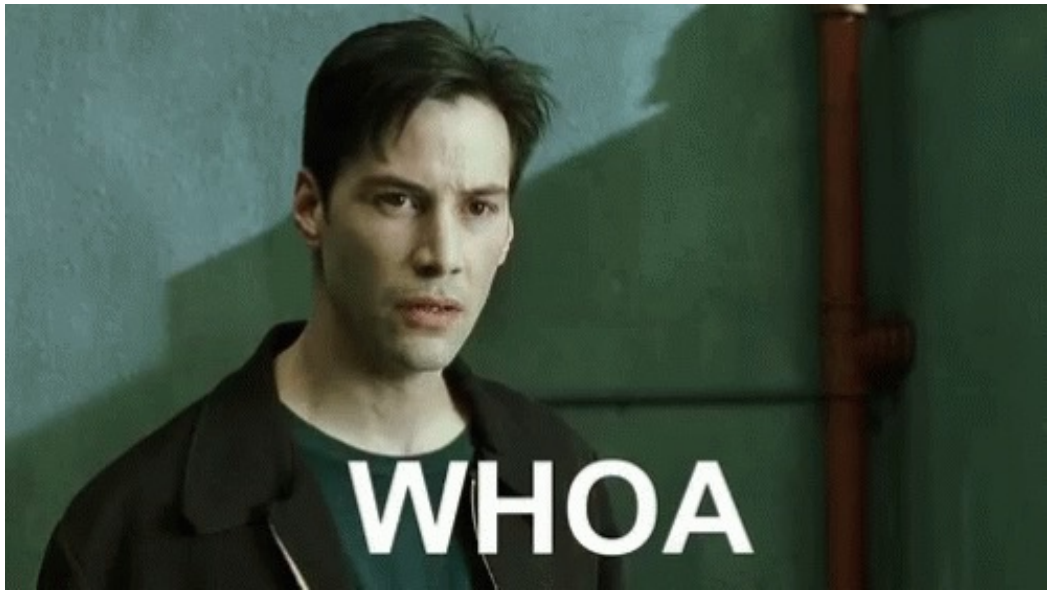
```python
def alignScore(S1, S2):
    '''Accepts two strings as inputs. Returns the maximum score
    needed to align the inputs.'''
    MATCH_REWARD = 1
    GAP_COST = -1 # insertions or deletions
    CHG_COST = -1
```

More sophisticated scoring systems: the scoring matrix

To this base

From this base

	A	T	G	C
A	5	-4	-1	-4
T	-4	5	-4	-1
G	-1	-4	5	-4
C	-4	-1	-4	5



Representing matrices with dictionaries

```
dnamat = {('A', 'A'): 5,  ('A', 'T'): -4, ('A', 'G'): -1,
          ('A', 'C'): -4, ('T', 'A'): -4, ('T', 'T'): 5,
          ('T', 'G'): -4, ('T', 'C'): -1, ('G', 'A'): -1,
          ('G', 'T'): -4, ('G', 'G'): 5,  ('G', 'C'): -4,
          ('C', 'A'): -4, ('C', 'T'): -1, ('C', 'G'): -4,
          ('C', 'C'): 5}

>>> dnamat[("A","T")]
-4
>>> dnamat[("A","G")]
-1
```

Worksheet

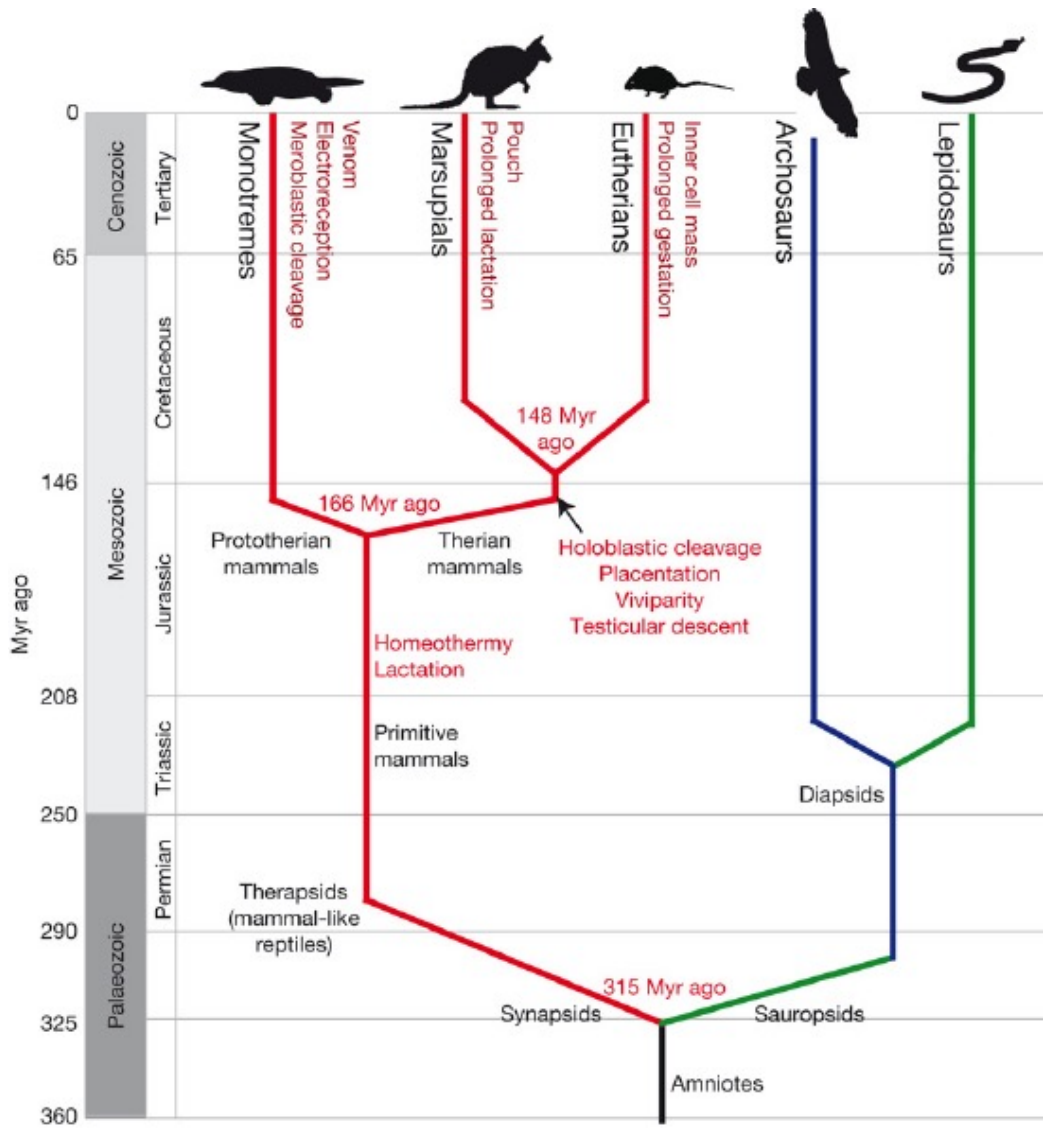
alignScore (matrix version)

Q

```
def alignScore(S1, S2, gap, submatD):
    '''Accepts two strings, a gap cost, and a dictionary as inputs.
    Returns the maximum score needed to align the input strings.'''
    if S1 == '':
        return gap * len(S2)
    elif S2 == '':
        return gap * len(S1)
    else:
        cvtCost = ???
        convert = ???
        delete  = gap + alignScore(S1[1:], S2, gap, submatD)
        insert  = gap + alignScore(S1, S2[1:], gap, submatD)
        return max(convert, delete, insert)
```

```
>>> alignScore("GCCTGG","ACCGGA",-4,dnamat)
11
```

Is the mammalian X homologous to the avian Z?



Nature 453, 175-183 fig1

Finding orthologs with `alignScore`

```
>>> prot1 =
'MMKTMSSGNCTLNVPAKNSYRMVVLGASRVGKSSIVSRFLNGRFEDQYTPTIEDFHRKVYNIRGDMYQLDILDTSGNH
PFPAMRRLSILTGDVFILVFSLDNRESFDEVKRLQKQILEVKSCLKNKTKESADLPMVICGNKNDHSEIYRKVRSDEGE
NLVSSDENCAYFEVSAKKNTNVDEMFYVLFSMAKLPHEMSPALHRKISIQYGDTFQQKSFRMRRVKDMDAYGMISPFAR
RPSVNSDLKYIKSKVLREGQSREREKCTIQ'

>>> prot2 =
'MTTIPRKGSSHLPGSLHTCKLKLQEDRRQQEKSVIAQPIFVFEKGEQTFKRPAEDTLYEAAEPECNGFPTKRVRSSSF
TFHITDSQSQGVRKNNVFMTSALVQSSVDIKSAEQGPVKHSKHVIRPAILQLPQARSCAKVRKTFGHKALESCKTKEKT
NNKISEGNSYLLSENLSRARISVQLSTNQDFLGATSVGCQPNEDKCSFKSCSSNFVFGENMVERVLGTQKLTQPQLEND
SYAKEKPFKSIPKFPVNFLSSRTDSIKNTSLIESAAAFSSQPSRKCLLEKIDVITGEETEHNVLKINCKLFIFNKTTQS
WIERGRGTLRLNDTASTDCGTLQSRLIMRNQGSLRLILNSKLWAQMKIQRANHKNVRITATDLEDYSIKIFLIQASAQD
TAYLYAAIHHRLVALQSFNKQRDVNQAESLSETAQQLNCESCDENEDDFIQVTKNGSDPSSWTHRQSVACS'


>>> memoAlignScore(prot1, prot2, -9, blosum62, {})
-1322
```

sequences

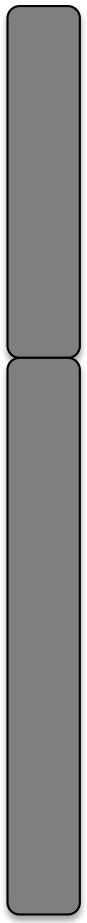
scoring matrix

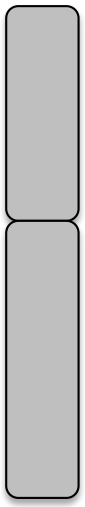
gap cost

memoD

Comparing chromosomes

Human X
156,000,000 bp

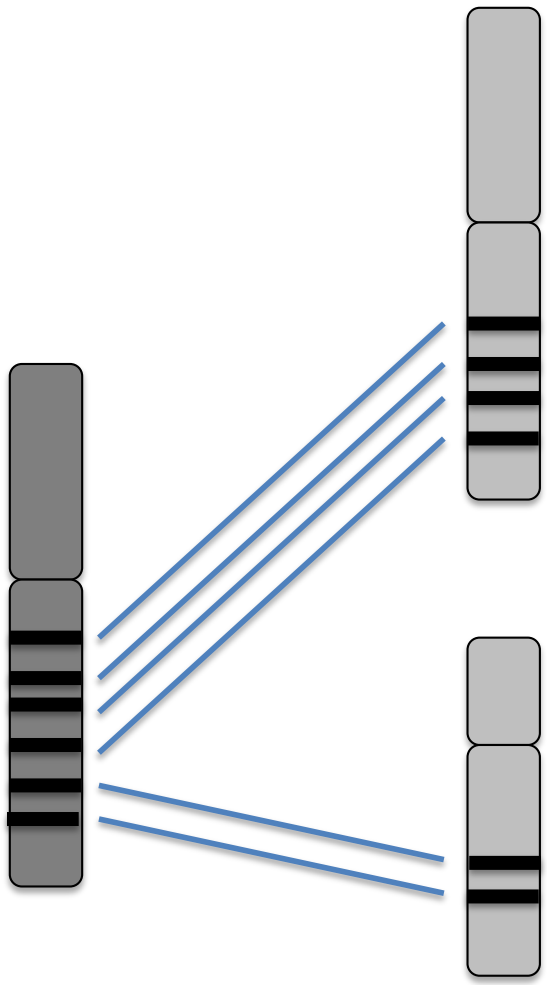




Chicken Z
82,000,000 bp

HW5 Preview

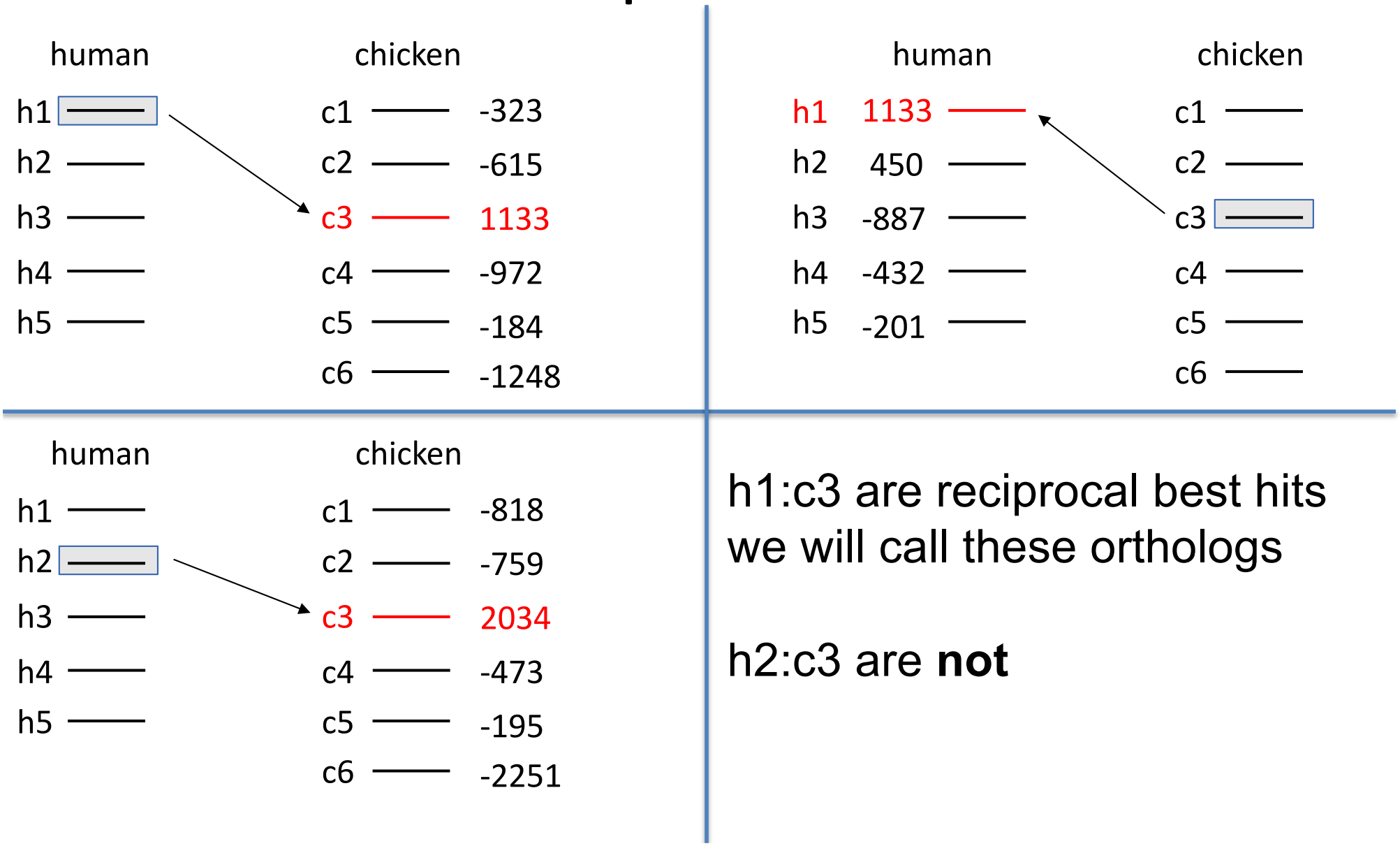
Chromosomes are too enormous and contain other types of mutations



We will find and compare orthologs instead!

HW5 Preview

Finding orthologs with `alignScore`: the best reciprocal hit method



h1:c3 are reciprocal best hits
we will call these orthologs

h2:c3 are **not**

Next time: returning the alignment

```
>>> alignScore("GCCTGG", "ACCGGA", -4, dnamatD)
11
```

`align_score` only returns score

```
>>> align("GCCTGG", "ACCGGA", -4, dnamatD)
[11, 'gCCtGG-', 'aCC-GGa']
```

`align` returns score **and** alignment

```
gCCtGG-
aCC-GGa
```

Reminder:

- Lecture feedback form (https://forms.gle/aPmkpXDUTp4Xo4CV7)