- Previous Lecture:
  - Image processing
    - 3-d array, computing with type `uint8`, vectorized code
  - Read 12.4 of textbook (image processing, type `uint8`)

- Today's Lecture:
  - Computing with characters (arrays of type `char`)
  - Review top-down design for program development
  - Linear search

- Announcements:
  - Project 4 due Monday at 11pm EDT
  - Consulting hours have resumed virtually
  - Work with course staff to review Prelim 1. Now is the time to firm up any loose foundation!

Text in programming

- We’ve seen text already
  - `fprintf('Hello world\n')`, `title('Click here')`, etc.
  - Time to dive into the details

Vocabulary:

- A single letter (or digit, or symbol, or space) is a “character”
- A sequence of characters is called a “string”
  - Could be a word, a sentence, gibberish

Text—sequences of characters often called strings—are important in computation

Numerical data is often encoded in strings. E.g., a file containing Ithaca weather data begins with the string

`W07629N4226`

meaning

Longitude:	76^{o} 29′ West
Latitude:	42^{o} 26′ North

We may need to grab hold of the substring `W07629`, convert `076` and `29` to the numeric values 76 and 29, and do some computation

Character array (an array of type `char`)

- We have used strings of characters in programs already:
  - `c= input('Give me a letter: ', 's')`
  - `msg= sprintf('Answer is %d', ans);`
- A string is made up of individual characters, so a string is a 1-d array of characters
- `'CS1112 rocks!'` is a character array of length 13; it has 7 letters, 4 digits, 1 space, and 1 symbol.

'C'	'S'	'1'	'1'	'1'	'2'	' '	'r'	'o'	'c'	'k'	's'	'!'

Row vector of length 13

- Can have 2-d array of characters as well

'C'	'S'	'1'	'1'	'1'	'2'
'r'	'o'	'c'	'k'	's'	'!'

2×6 matrix

A text sequence is a vector (of characters)

Vectors

- Assignment

```
v= [7, 0, 5];
```

- Indexing

```
x= v(3);     % x is 5
v(1)= 1;     % v is [1 0 5]
w= v(2:3);   % w is [0 5]
```

- : notation

```
v= 2:5;      % v is [2 3 4 5]
```

- Appending

```
v= [7 0 5];
v(4)= 2;     % v is [7 0 5 2]
```

- Concatenation

```
v= [v [4 6]];
         % v is [7 0 5 2 4 6]
```

Strings

- Assignment

```
s= ['h','e','l','l','o'];
                % formal
s= 'hello';   % shortcut
```

- Indexing

```
c= s(2);     % c is 'e'
s(1)= 'J';   % s is 'Jello'
t= s(2:4);   % t is 'ell'
```

- : notation

```
s= 'a':'g'; % s is 'abcdefg'
```

- Appending

```
s= 'duck';
s(5)= 's';   % s is 'ducks'
```

- Concatenation

```
s= [s ' quack'];
           % s is 'ducks quack'
```

Syntax: Single quotes enclose char arrays in Matlab

Anything enclosed in single quotes is a string (*even if it looks like something else*)

`'100'`	is a character array (string) of length 3
`100`	is a numeric value
`'pi'`	is a character array of length 2
`pi`	is the built-in constant 3.14159…
`'x'`	is a character (vector of length 1)
`x`	may be a variable name in your program

Types so far: `char`, `double`, `logical`

```
a= 'CS1'
a= ['C','S','1']
```

a is a 1-d array with type `char` elements. Often called a *string*; NOT the same as a *new* type in Matlab 2017+ called `string`.

a	'C'	'S'	'1'

```
b= [3 9]
```

b is a 1-d array with type `double` elements. `double` is the default type for numbers in Matlab. We call b a “numeric array”

```
c= uint8(b)
```

c is a 1-d array with type `uint8` elements. We call c a “uint8 array”

```
d= rand() > .5
```

d is a scalar of the type `logical`. We call d a “Boolean value”

Basic (simple) types in MATLAB

- E.g., `char`, `double`, `unit8`, `logical`
- Each uses a set amount of memory
  - Each `uint8` value uses 8 bits (=1 byte)
  - Each `double` value uses 64 bits (=8 bytes)
  - Each `char` value uses 16 bits (=2 bytes)
  - Use function `whos` to see memory usage by variables in workspace
- Can easily determine amount of memory used by a simple array (array of a basic type, where each component stores one simple value)
- Next lecture: Special arrays where each component is a container for a collection of values

Self-check

What is the value of `substr`?

```
str = 'My hovercraft is full of eels.';
substr = str(19:length(str)-2);
```

A `'ll of eels'`

B `'ull of eel'`

C `['o', 'f', 'e', 'e']`

D `[19 20 … 28]`

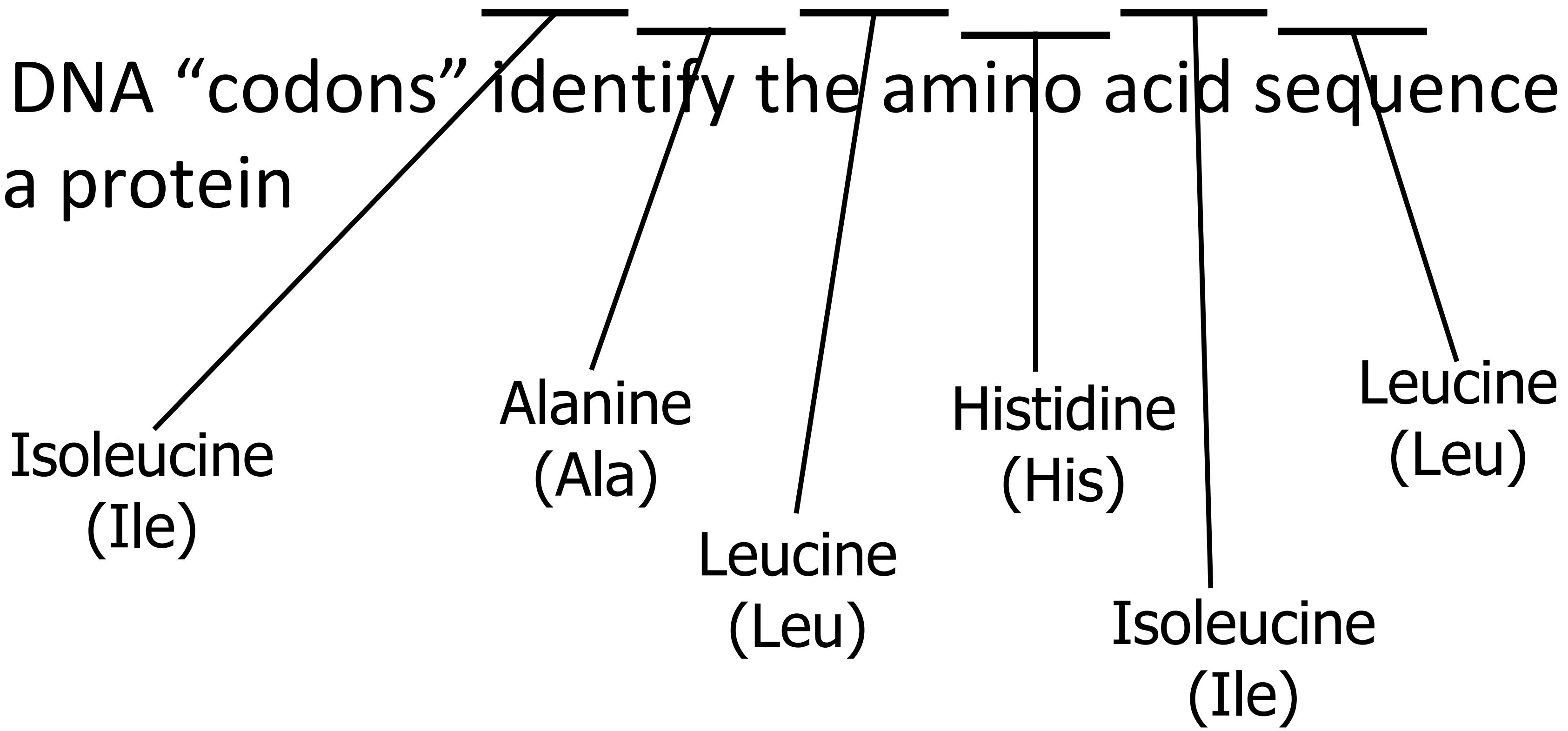
E *None of the above*

Working with gene data → compute on text data

- A gene is a DNA fragment that codes for a protein, e.g.,
  ATCGCTTTGCACATTCTA...
- 3-letter DNA "codons" identify the amino acid sequence that defines a protein

Working with gene data → compute on text data

- A gene is a DNA fragment that codes for a protein, e.g.,
  ATCGCTTTGCACATTCTA…
- 3-letter DNA "codons" identify the amino acid sequence that defines a protein



The Codon Dictionary

Index	Amino Acid	Mnemonic	DNA Codons
1	Alanine	Ala	GCT GCC GCA GCG
2	Arginine	Arg	CGT CGC CGA CGG AGA AGG
3	Asparagine	Asn	AAT AAC
4	Aspartic Acid	Asp	GAT GAC
5	Cysteine	Cys	TGT TGC
6	Glutamic Acid	Glu	CAA CAG
7	Glutamine	Gln	GAA GAG
8	Glycine	Gly	GGT GGC GGA GGG
9	Histidine	His	CAT CAC
10	Isoleucine	Ile	ATT ATC ATA
11	Leucine	Leu	CTT CTC CTA CTG TTA TTG
12	Lysine	Lys	AAA AAG
13	Methionine	Met	ATG
14	Phenylalanine	Phe	TTT TTC
15	Proline	Pro	CCT CCC CCA CCG
16	Serine	Ser	TCT TCC TCA TCG AGT AGC
17	Threonine	Thr	ACT ACC ACA ACG
18	Tryptophan	Trp	TGG
19	Tyrosine	Tyr	TAT TAC
20	Valine	Val	GTT GTC GTA GTG

Visualize distribution of amino acid in a protein

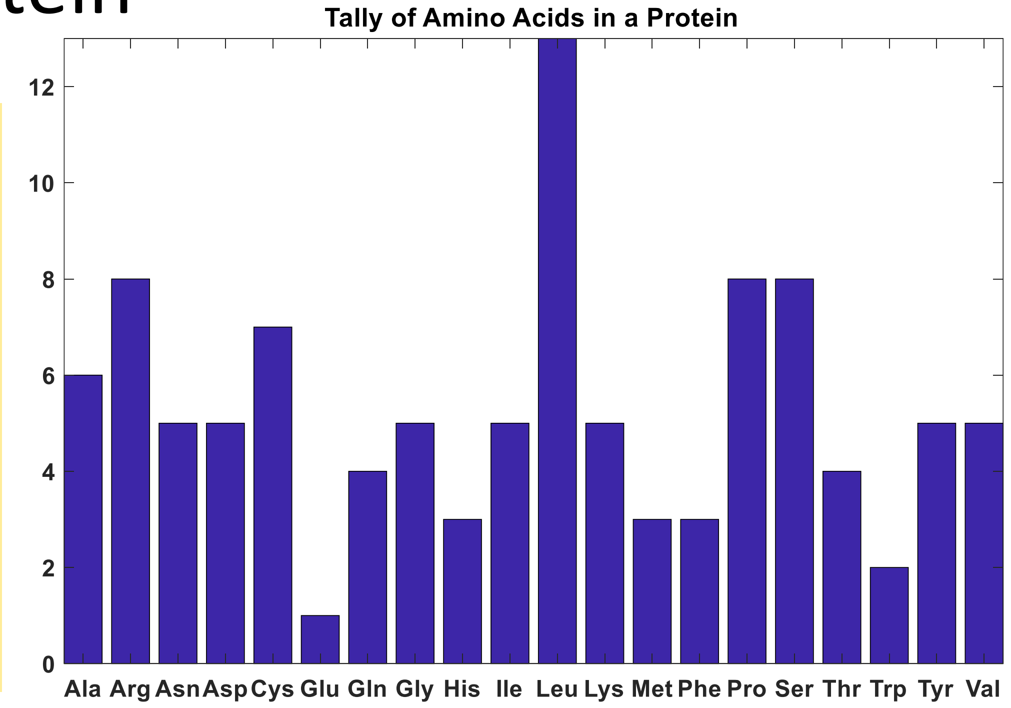
- Given a gene sequence defining a protein

  TTCGGGAGCCTGGGCGTTACG…

- Make histogram showing counts of amino acids that make up the protein

Compute with text data!

- Create `char` arrays
- Obtain subarrays (each a 3-letter codon)
- Search for and compare subarrays
- Do tally, draw histogram



Program sketch

- Given a dna sequence representing a protein

- For each codon (subvector of 3 chars)
  - Use codon dictionary to determine which amino acid the codon represents (get the 3-letter mnemonic)

- Tally the counts of the 20 amino acids

- Draw bar chart

See Insight §9.1. Here in lecture we extend the two functions for searching char arrays.

```
% dna sequence encoding protein
p= ['TTCGGGAGCCTGGGCGTTACGTTAATGAAA' ...
    'ATATGTACCAACGACAATGACATTGAAAAC'];
```

p is a 1-d array

Program sketch

- Given a ✓ dna sequence representing a protein
- For each codon (subvector of 3 chars)
    - Use codon dictionary to determine which amino acid the codon represents (get the 3-letter mnemonic)
- Tally the counts of the 20 amino acids
- Draw bar chart

```
% dna sequence encoding protein
p= ['TTCGGGAGCCTGGGCGTTACGTTAATGAAA' ...
    'ATATGTACCAACGACAATGACATTGAAAAC'];


for k= 1:3:length(p)-2
   codon= p(k:k+2); % length 3 subvector

   % Search codon dictionary to find
   % the corresponding amino acid name


end
```

Start index: k

End index: k + length of codon - 1

Treat as an independent task
to be written as a function

```
function a = getMnemonic(s)
% s is length 3 row vector of chars
% If s is codon of an amino acid then
% a is the mnemonic of that amino acid

% Search for s in codon dictionary C
C= ['GCT Ala'; ...
    'GCC Ala'; ...
    'GCA Ala'; ...
    'GCG Ala'; ...
    'CGT Arg'; ...
    'CGC Arg'; ...
    'CGA Arg'; ...
    'CGG Arg'; ...
    'AGA Arg'; ...
    'AGG Arg';
```

C is a <u>2-d array</u> of chars

'G'	'C'	'T'	' '	'A'	'l'	'a'
'G'	'C'	'C'	' '	'A'	'l'	'a'
'G'	'C'	'A'	' '	'A'	'l'	'a'
'G'	'C'	'G'	' '	'A'	'l'	'a'
'C'	'G'	'T'	' '	'A'	'r'	'g'
'C'	'G'	'C'	' '	'A'	'r'	'g'

```
function a = getMnemonic(s)
⋮
% Given C, the 2-d char array dictionary
% Search it to find string s


r= 1;
while     strcmp(s, C(r, 1:3))==false
   r= r + 1;
end

a= C(r, 5:7);
```

Compares two char vectors.
Returns true if they are identical;
otherwise returns false.

Assumes s is in C! If s not found → ERROR

'G'	'C'	'T'	' '	'A'	'l'	'a'
'G'	'C'	'C'	' '	'A'	'l'	'a'
'G'	'C'	'A'	' '	'A'	'l'	'a'
'G'	'C'	'G'	' '	'A'	'l'	'a'
'C'	'G'	'T'	' '	'A'	'r'	'g'
'C'	'G'	'C'	' '	'A'	'r'	'g'

```
function a = getMnemonic(s)
⋮
% Given C, the 2-d char array dictionary
% Search it to find string s



r= 1;
while      strcmp(s, C(r, 1:3))==false
   r= r + 1;
end

a= C(r, 5:7);
```

Modify function so that a gets empty char array if s not found

'G'	'C'	'T'	' '	'A'	'l'	'a'
'G'	'C'	'C'	' '	'A'	'l'	'a'
'G'	'C'	'A'	' '	'A'	'l'	'a'
'G'	'C'	'G'	' '	'A'	'l'	'a'
'C'	'G'	'T'	' '	'A'	'r'	'g'
'C'	'G'	'C'	' '	'A'	'r'	'g'

```
function a = getMnemonic(s)
⋮
% Given C, the 2-d char array dictionary
% Search it to find string s
a= '';
nr= size(C, 1);
r= 1;
while r<=nr && strcmp(s, C(r, 1:3))==0
    r= r + 1;
end
if r<=nr
    a= C(r, 5:7);
end
```

If s not in C then a gets empty char array

'G'	'C'	'T'	' '	'A'	'l'	'a'
'G'	'C'	'C'	' '	'A'	'l'	'a'
'G'	'C'	'A'	' '	'A'	'l'	'a'
'G'	'C'	'G'	' '	'A'	'l'	'a'
'C'	'G'	'T'	' '	'A'	'r'	'g'
'C'	'G'	'C				

See getMnemonic.m

```
% dna sequence encoding protein
p= ['TTCGGGAGCCTGGGCGTTACGTTAATGAAA' ...
    'ATATGTACCAACGACAATGACATTGAAAAC'];

for k= 1:3:length(p)-2
   codon= p(k:k+2); % length 3 subvector

   % Search codon dictionary to find
   % the corresponding amino acid name
   mnem= getMnemonic(codon);




end
```

Program sketch

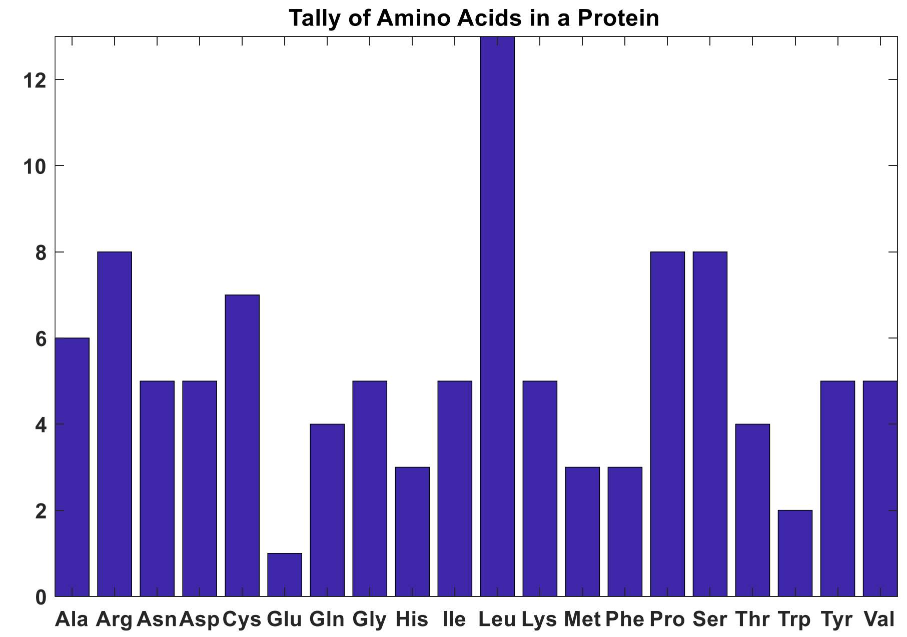
- ✓ Given a dna sequence representing a protein
- ✓ For each codon (subvector of 3 chars)
    - Use codon dictionary to determine which amino acid the codon represents (get the 3-letter mnemonic)
- Tally the counts of the 20 amino acids
- Draw bar chart

```
% dna sequence encoding protein
p= ['TTCGGGAGCCTGGGCGTTACGTTAATGAAA' ...
    'ATATGTACCAACGACAATGACATTGAAAAC'];

for k= 1:3:length(p)-2
   codon= p(k:k+2); % length 3 subvector
   mnem= getMnemonic(codon);
   % Tally: build histogram data




end
```



```
% dna sequence encoding protein
p= ['TTCGGAGCCTGGGCGTTACGTTAATGAAA' ...
    'ATATGTACCAACGACAATGACATTGAAAAC'];

count= zeros(1,20); % to store tallies

for k= 1:3:length(p)-2
   codon= p(k:k+2); % length 3 subvector
   mnem= getMnemonic(codon);
   % Tally: build histogram data
   ind= getAAIndex(mnem);
   count(ind)= count(ind) + 1;
end
bar(1:20, count)   % Draw bar chart
```

```
function ind = getAAIndex(aa)
% Returns index of amino acid named by char vector aa.
% If aa does not name an amino acid, throw an error.
```

Display an error message and STOP program execution. (Not just a print statement.)
Use built-in function **`error`**.
See **`getAAIndex.m`**

Syntax: **error( )**

message to display

```
% dna sequence encoding protein
p= ['TTCGGAGCCTGGGCGTTACGTTAATGAAA' ...
    'ATATGTACCAACGACAATGACATTGAAAAC'];

count= zeros(1,20); % to store tallies

for k= 1:3:length(p)-2
   codon= p(k:k+2); % length 3 subvector
   mnem= getMnemonic(codon);
   % Tally: build histogram data
   ind= getAAIndex(mnem);
   count(ind)= count(ind) + 1;
end
bar(1:20, count)  % Draw bar chart
```

See **aminoAcidCounts.m**

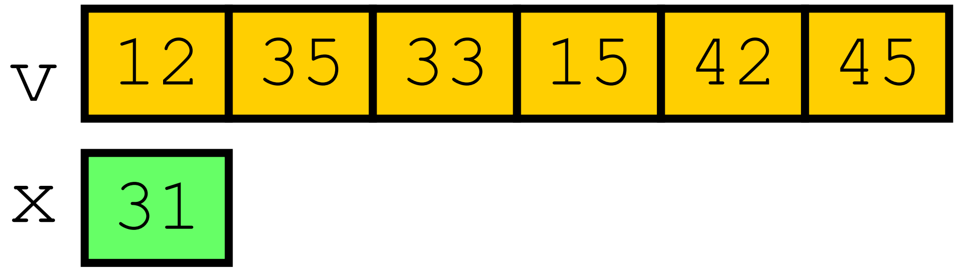
In addition to type **`char`**, we discussed …

- Top-down design in program development—decompose the problem and then build the program one subproblem (one part, one refinement) at a time
- Search: Linear Search Algorithm

```
k= 1
while  k is valid and
             item at k does not match search target

          k= k + 1
end
```

```
% Linear Search
% f is index of first occurrence
%   of value x in vector v.
% f is -1 if x not found.
k= 1;
while  k<=length(v) && v(k)~=x
    k= k + 1;
end
if  k>length(v)
    f= -1; % signal for x not found
else
    f= k;
end
```



```
% Linear Search
% f is index of first occurrence
%   of value x in vector v.
% f is -1 if x not found.
k= 1;
while  k<=length(v) && v(k)~=x
    k= k + 1;
end
if  k>length(v)
    f= -1; % signal for x not found
else
    f= k;
end
```

A. squared

B. doubled

C. the same

D. halved

Suppose another vector is twice as long as v. The expected “effort” required to do a linear search is …