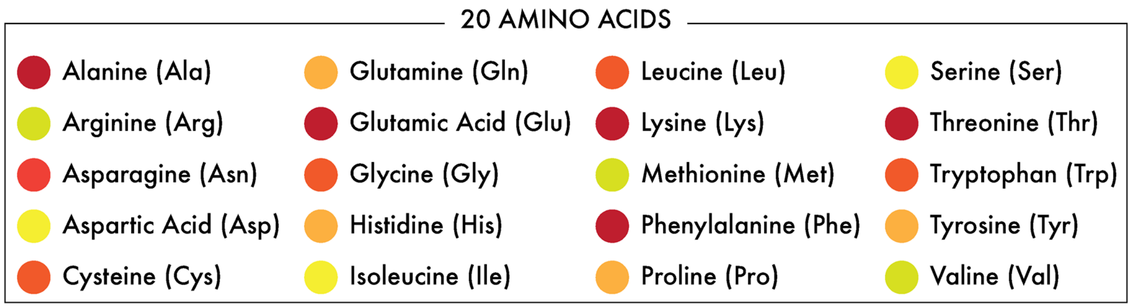
Lesson 7

August 7, 2020

1 Lesson 7: Build Your Own Protein I

1.1 The Genetic Code

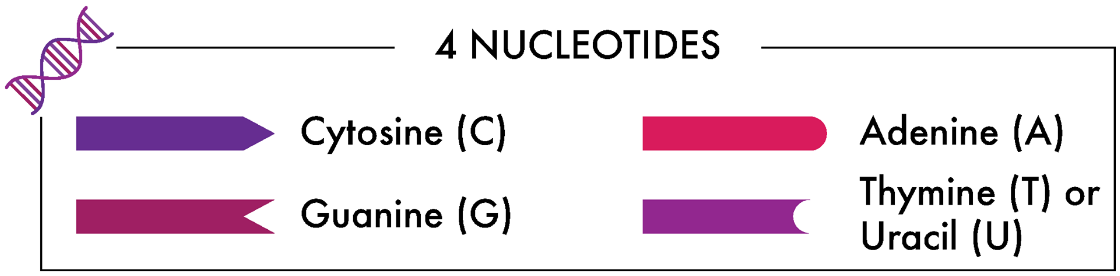
So we've talked about how our genes contribute to our phenotype. But what exactly are genes? Genes, as we've mentioned, are made up of DNA and provide the information for our cells to carry out important biological processes that ultimately lead to phenotype. Those processes depend on molecules called proteins. The DNA provides instructions for building proteins, which are made up of essential building blocks called amino acids.



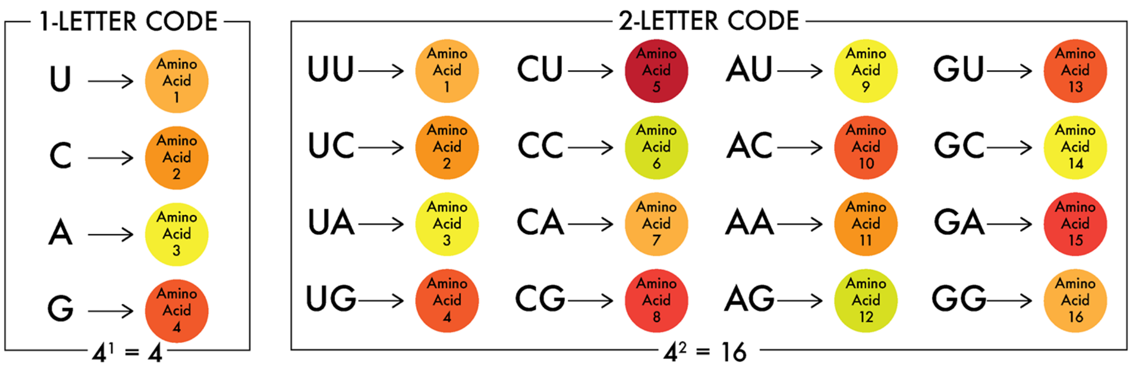
20 AMINO ACIDS

● Alanine (Ala)	● Glutamine (Gln)	● Leucine (Leu)	● Serine (Ser)
● Arginine (Arg)	● Glutamic Acid (Glu)	● Lysine (Lys)	● Threonine (Thr)
● Asparagine (Asn)	● Glycine (Gly)	● Methionine (Met)	● Tryptophan (Trp)
● Aspartic Acid (Asp)	● Histidine (His)	● Phenylalanine (Phe)	● Tyrosine (Tyr)
● Cysteine (Cys)	● Isoleucine (Ile)	● Proline (Pro)	● Valine (Val)

Almost all proteins are made up of combinations and permutations of the same 20 amino acids. This is where the code of DNA comes in. DNA is composed of 4 different molecules called nucleotides or bases: adenosine, thymine, cytosine, and guanine. We abbreviate these as A, T, C, and G, respectively.



An important question is immediately apparent: how many ways can 4 base pairs code 20 unique amino acids? The answer is in combinations. Let's do some quick math to figure out how many letters in a row you need to code for 20 amino acids.



But wait - we know there are 20 amino acids! A 1-letter code only created 4 amino acids and a 2-letter code only created 16 amino acids. Can you determine how many letters are necessary to create all 20 amino acids?

Note that in Python, we use "**" as an exponent symbol (i.e. to calculate 3 to the 8th power [3^8], we would type 3**8).

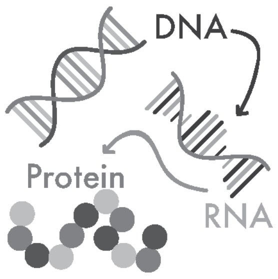
[1]:
```python
print('A 1-letter code:', 4**1)
print('A 2-letter code:', 4**2)
print('A 3-letter code:', 4**3)
```

```
('A 1-letter code:', 4)
('A 2-letter code:', 16)
('A 3-letter code:', 64)
```

That's right, there are 64 3-letter codes, but only 20 of them code useful amino acids that form proteins in the human body!

1.2 The Central Dogma

This triplet genetic code is actually conferred in the biology of using DNA to make proteins through an intermediate, RNA; RNA, or ribonucleic acid, is composed of three of the same nucleotides as DNA: adenosine, cytosine, and guanine, and with uracil replacing thymine. DNA is copied into RNA in a process called transcription. This is easy to remember, since DNA and RNA are "written" in the same "language" of nucleotides. RNA is then used as a template to make a protein in a process called translation. This makes sense, as RNA and proteins are "written" in different "languages" (nucleotides and amino acids, respectively). The transcription of DNA into RNA, which can then be translated into proteins is called The Central Dogma of biology.



Quick Quiz: - How many nucleotides code for a single amino acid? 3 nucleotides : 1 amino acid (a code of 3 nucleotides allows for 64 possible combinations, more than enought to encode 20 unique amino acids) - What amino acid does the RNA sequence GAA code for? Glu (Glutamine) (looking it up in the genetic code, GAA codes Glu) - What is transcription? The process of copying the information in DNA into RNA

1.3 Lists

So far, we have used variables to store one value at a time. For example, we could have stored the name of one amino acid as a variable:

```
[2]: my_amino_acid = 'methionine'
     print(my_amino_acid)
```

```
methionine
```

But as you just learned, there are 20 different amino acids. So if we wanted to store all 20 amino acids as variables, we would have to create 20 different variables (e.g., my_amino_acid_1, my_amino_acid_2, ..., my_amino_acid_20). An easier way to store many valuables is, quite intuitively, in a list. A list is simply a changable, ordered sequence of elements. Just like strings are defined as ordered characters between quotes "", lists are defined as comma-delimited elements between square brackets []. (Lists can also store any data type or combination of data types — for example, strings and floats can be in a single list together.) We can then store all 20 amino acids as a list:

```
[3]: # Store all 20 amino acids in a list
     amino_acids = ['alanine', 'arginine', 'asparagine', 'aspartic acid',␣
      ↪'cysteine', 'glutamic acid', 'glutamine', 'glycine', 'histidine',␣
      ↪'isoleucine', 'leucine', 'lysine', 'methionine', 'phenylalanine', 'proline',␣
      ↪'serine', 'threonine', 'tryptophan', 'tyrosine', 'valine']
     print(amino_acids)
```

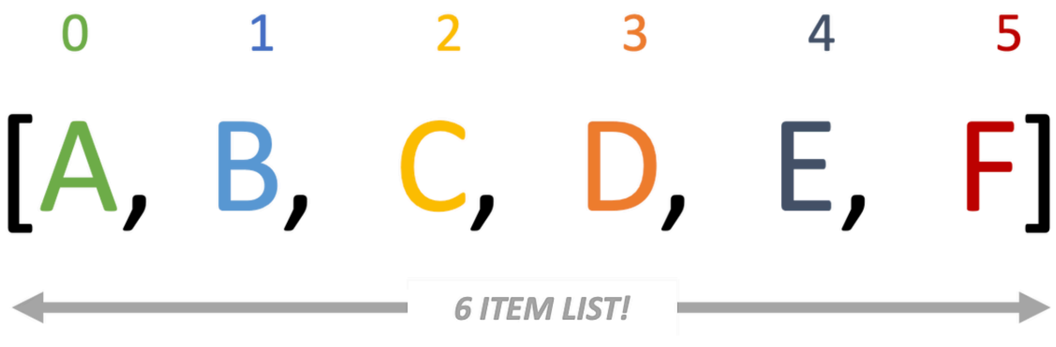
```
['alanine', 'arginine', 'asparagine', 'aspartic acid', 'cysteine', 'glutamic
acid', 'glutamine', 'glycine', 'histidine', 'isoleucine', 'leucine', 'lysine',
'methionine', 'phenylalanine', 'proline', 'serine', 'threonine', 'tryptophan',
'tyrosine', 'valine']
```

Let's discover some other things about lists. Type the following into the terminal and see what happens!

```
[4]: amino_acids = ['alanine', 'arginine', 'asparagine', 'aspartic acid',␣
      ↪'cysteine', 'glutamic acid', 'glutamine', 'glycine', 'histidine',␣
      ↪'isoleucine', 'leucine', 'lysine', 'methionine', 'phenylalanine', 'proline',␣
      ↪'serine', 'threonine', 'tryptophan', 'tyrosine', 'valine']
     print(amino_acids[0])
     print(len(amino_acids))
```

```
alanine
20
```

We have said that a list is an ordered sequence of elements. This ordering is useful, in that we can access the first, fifth, or last element in the list very quickly as well as determine the length of the list.



The number in the list that an element occurs is called that element's index. To print any element in a list, use the syntax my_list[index]. But be careful: Python lists are zero-indexed! That is, the first element in the list has index 0, and the fifth element in the list has index 4. So, to print the first amino acid in our protein and the length of that list:

Print the last amino acid in the following amino acid list WITHOUT COUNTING! Hint: remember that the list is zero-indexed and that the len() formula gives you the total length of the list

[7]:
```
my_protein = ['methionine','valine','leucine','serine', 'proline', 'alanine',␣
 ↪'lysine', 'threonine','asparagine', 'valine','lysine', 'tryptophan',␣
 ↪'glycine','lysine', 'valine', 'glycine','alanine']
last_item = my_protein[ len(my_protein) - 1 ]
print(last_item)
```

```
alanine
```

We started by retrieving the length of the list, and remembered that lists are zero-indexed (e.g., if there are 10 items in a list the last item will be stored here list[9]). Then stored that in a variable and printed to the console

1.4 For loops

Let's consider a list of 10 numbers, 0 to 9:

[8]:
```
my_list = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
print(my_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

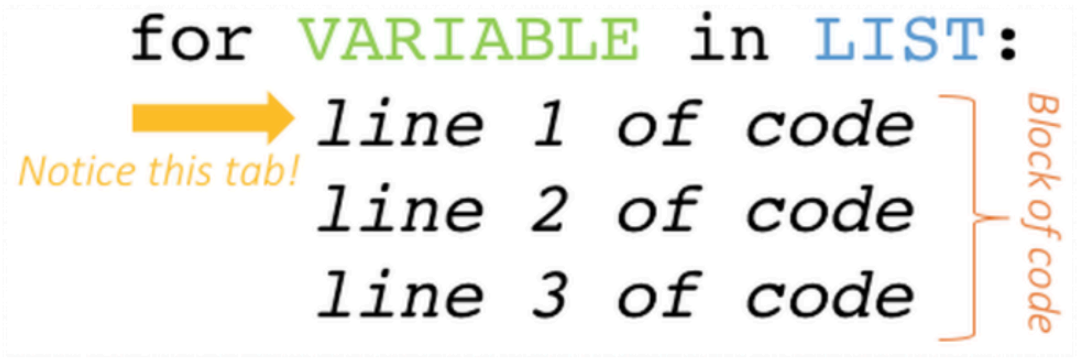
It would be easier if we didn't have to type out each number (computers should be able to count, shouldn't they?). There is a convenient function, list(range()), that allows us to create a list containing every number between a specified start and stop.

[9]:
```
my_list = list(range(0, 10))
print(my_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Now that we have lists of numbers, what if we wanted to execute the same function to every element in the list? For example, we can add 5 to each element in my_list with a simple operation. In computer science, a for loop allows code to be executed repeatedly (in this case, the code we want to execute repeatedly adds 5 to each element in the list).

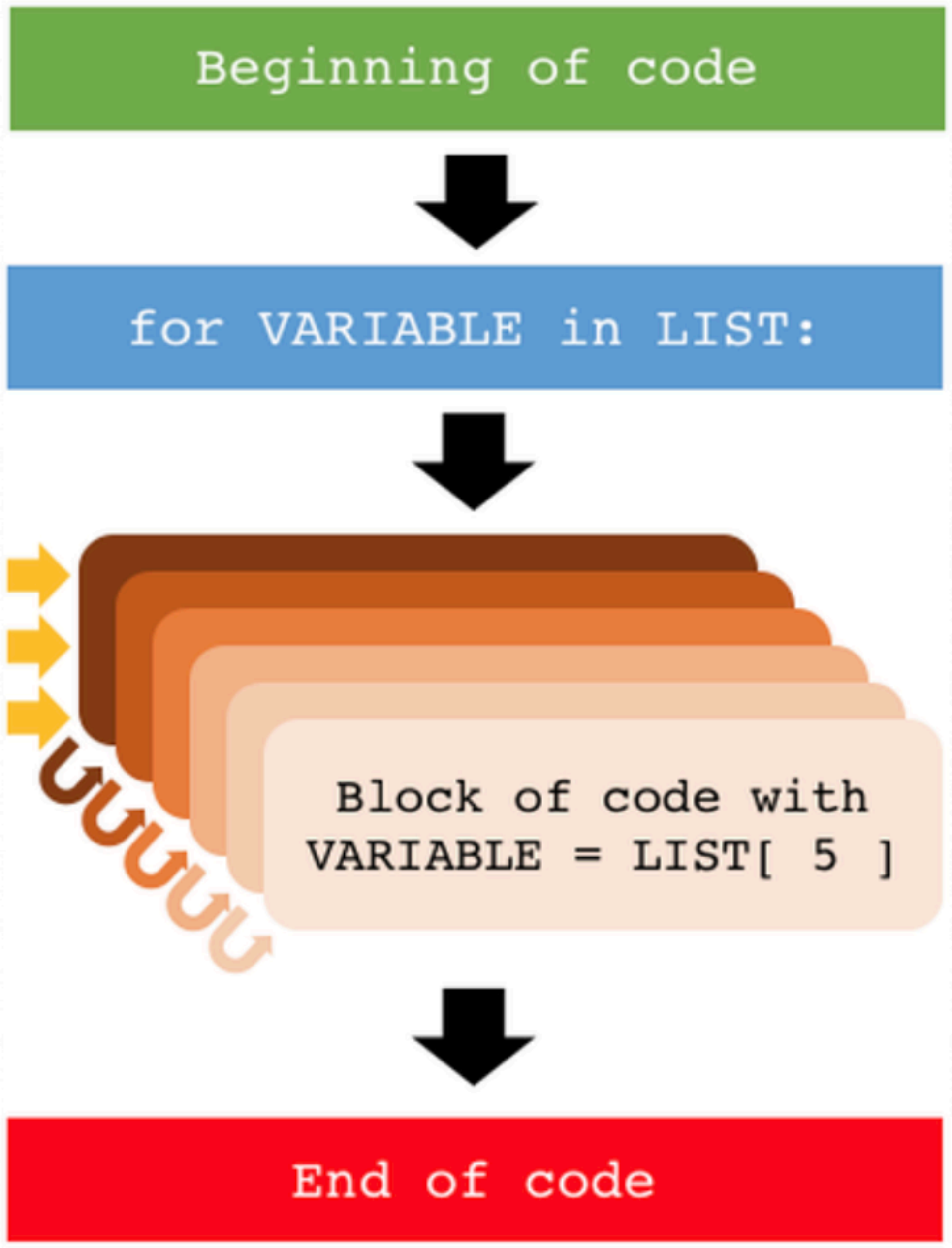
Every for loop starts with the same syntax:



```
[10]: my_list = list(range(0, 10))
      for my_number in my_list:
          print(my_number + 10)
```

```
10
11
12
13
14
15
16
17
18
19
```

Predict what you think the output of this code will be and then check



1.5 For Loops & If Statements

Now that we know what a for loop is, how can we use a for loop to better understand our protein? For example, the amino acid lysine is positively charged, so to get a sense of the overall positive charge of a protein, we might want to know the number of lysines it contains. We can use a number of tricks that we've already learned to count the number of lysines in our protein.

To do this, we can "loop through" each element in our list and check if each element (amino acid) is a lysine or not. We use a variable to keep track of the number of lysines that we encounter. Then, we can update this count as we move through the loop, adding 1 each time we observe a lysine.

With that basic approach in mind, there are a couple of ways to structure a for loop.

First, we can use range() to create a list of numbers such that we can index into each value in the protein in order. That is, we want to create a list of numbers from 0 to the length of the protein. These numbers can then serve as indices to access each amino acid in our protein, one by one. So all together, here's what it looks like (don't worry if it seems confusing, we'll step through it together below):

[12]:
```
my_protein = ['methionine','valine','leucine','serine', 'proline', 'alanine',␣
↪'lysine', 'threonine', 'asparagine','valine','lysine','alanine', 'alanine',␣
↪'tryptophan', 'glycine','lysine', 'valine', 'glycine','alanine']

# Way 1
lysine_count = 0

# Since we haven't looked at the protein yet, we have observed 0 lysines so far
for i in range(0,len(my_protein)):
    amino_acid = my_protein[i]
    if amino_acid == 'lysine':
        lysine_count = lysine_count + 1
# For each lysine that we observe, we add 1 to the value of the variable␣
↪keeping track of the number of lysines in the protein

print('The number of lysines in our protein is', lysine_count)
```

```
('The number of lysines in our protein is', 3)
```

Okay, so what's happening here? We create a variable called lysine_count, and set its initial value to 0, since we haven't encountered any lysines in our protein so far. Now comes the for loop. Here, we create another variable, i, that will eventually hold the indexing value for each amino acid in our protein list. We create a list of these indices using range() for the entire length of our protein list (remember 0 indexing!). Now that we have these indices, we step into the body of the for loop. The first time we enter the for loop, the variable i takes on the value of 0 (the first value of the index list). So, when we index into our protein list with i = 0, we get the first element of the list (methionine), stored into a third variable, amino_acid. Then, we can use an if statement to check if the first amino acid is a lysine (remember that == is how we check if something is equal to another value). If the amino acid is a lysine, then we want to update lysine_count by adding 1 to our running count. Because our first amnio acid is a methionine, our count remains at 0. And then the loop starts all over again, this time with i = 1. The loop will continue in this manner until we reach i = 18, the very end of our protein list.

If this still seems a little confusing, try what programmers do all the time to figure out how code works: insert print statements into the body of the for loop to print out the value of variables (maybe try i, amino_acid, or lysine_count). If you get errors doing this, try moving the print statments to other parts of the loop!

But that's only one way to skin the cat. We can also side-step using indices by looping directly through the amino acids in the protein:

1.6 Checkpoint

Use a for loop to count the number of alanines in the protein

```
[13]: my_protein = ['methionine','valine','leucine','serine', 'proline', 'alanine',␣
       ↪'lysine', 'threonine', 'asparagine','valine','lysine','alanine', 'alanine',␣
       ↪'tryptophan', 'glycine','lysine', 'valine', 'glycine','alanine']

      alanine_count = 0

      # Since we haven't looked at the protein yet, we have observed 0 lysines so far
      for i in range(0,len(my_protein)):
          amino_acid = my_protein[i]
          if amino_acid == 'alanine':
              alanine_count = lysine_count + 1

      print('The number of alanines in our protein is', lysine_count)
```

```
('The number of alanines in our protein is', 3)
```

We started by creating a variable called “counter”. This is a VERY common. Then every time the loop looked at an item we compared it to ‘alanine’. If it matched then we added 1 to the counter. After we looped through the list we printed the result.

1.7 Summary

In this lesson, we’ve learned about the following:

Genetic Code

Central Dogma

Lists

For Loops

For Loops & If Statements