

Python Class 9

Making a recipe class, a todo list class, saving data

```
1  def add(a,b):  
2      return a + b  
3  
4  def square(a):  
5      return a * a  
6  
7  def half(a):  
8      return a/2  
9  
10 x = add(5,10)  
11 print(x)  
12  
13 x = add(5,square(7))  
14 print(x)  
15  
16 x = add(5,half(square(7)))  
17 print(x)  
18  
19 x = square(half(add(50,60)))  
20 print(x)
```

The diagram illustrates the execution flow of the code. A red arrow points from the `return a + b` statement on line 2 to the `add(5,10)` call on line 10. Another red arrow points from the `x = add(5,10)` call on line 10 to the `x = add(5,square(7))` call on line 13. A third red arrow points from the `x = add(5,square(7))` call on line 13 to the `x = add(5,half(square(7)))` call on line 16. A fourth red arrow points from the `x = add(5,half(square(7)))` call on line 16 to the `x = square(half(add(50,60)))` call on line 19. The `return a + b` statement on line 2 and the `add(5,10)` call on line 10 are highlighted with a cyan box.

```
1  def add(a,b):
2      return a + b
3
4  def square(a):
5      return a * a
6
7  def half(a):
8      return a/2
9
10 x = add(5,10)
11 print(x)
12
13 x = add(5,square(7))
14 print(x)
15
16 x = add(5,half(square(7)))
17 print(x)
18
19 x = square(half(add(50,60)))
20 print(x)
```

The diagram illustrates the execution flow of the code. A blue arrow points from the function call `add(5,square(7))` on line 13 to the `def add(a,b):` definition on line 1. Another blue arrow points from the `return a + b` statement on line 2 back to the call on line 13. A red arrow points from the same call on line 13 to the `def square(a):` definition on line 4. Another red arrow points from the `return a * a` statement on line 5 back to the call on line 13. A green box highlights the call `add(5,square(7))` on line 13.

```
1  def add(a,b):  
2      return a + b  
3  
4  def square(a):  
5      return a * a  
6  
7  def half(a):  
8      return a/2  
9  
10 x = add(5,10)  
11 print(x)  
12  
13 x = add(5,square(7))  
14 print(x)  
15  
16 x = add(5, half(square(7)))  
17 print(x)  
18  
19 x = square(half(add(50,60)))  
20 print(x)
```

The diagram illustrates the execution flow of the code. It shows three function definitions: `add(a,b)`, `square(a)`, and `half(a)`. The `add` function is called three times: `add(5,10)` on line 10, `add(5, square(7))` on line 13, and `add(5, half(square(7)))` on line 16. The `square` function is called on line 13 and line 16. The `half` function is called on line 16 and line 19. The `add` function's return value is assigned to `x` in each case. The `print` statement on line 17 prints the value of `x` after the third call to `add`. The `print` statement on line 20 prints the value of `x` after the final call to `square`.

```
1  def add(a,b):
2      return a + b
3
4  def square(a):
5      return a * a
6
7  def half(a):
8      return a/2
9
10 x = add(5,10)
11 print(x)
12
13 x = add(5,square(7))
14 print(x)
15
16 x = add(5, half(square(7)))
17 print(x)
18
19 x = square(half(add(50,60)))
20 print(x)
```

The diagram illustrates the execution flow of the provided Python code. Red arrows trace the return values from the `square(7)` call in line 13 back to its definition in line 5, and from the `half(square(7))` call in line 16 back to its definition in line 8. Blue arrows trace the return values from the `add(5,10)` call in line 10 back to its definition in line 2, and from the `half(add(50,60))` call in line 19 back to its definition in line 8. A green box highlights the `half(square(7))` expression in line 16, indicating the current state of the code.

Class Practice 2

- Make a recipe class.
- What variables do we need?
- What functions do we need?



Using `__repr__` (self)

- `__repr__()` and `__str__` are very similar functions.
- They both take `self` and return a string version.
- If you call `print(obj)`, the computer will use `obj.__str__()`
- If you call `print(objlist)`, the computer will use `__repr__()`


Class Practice 2

- Make a recipe class.
- What variables do we need?
- What functions do we need?



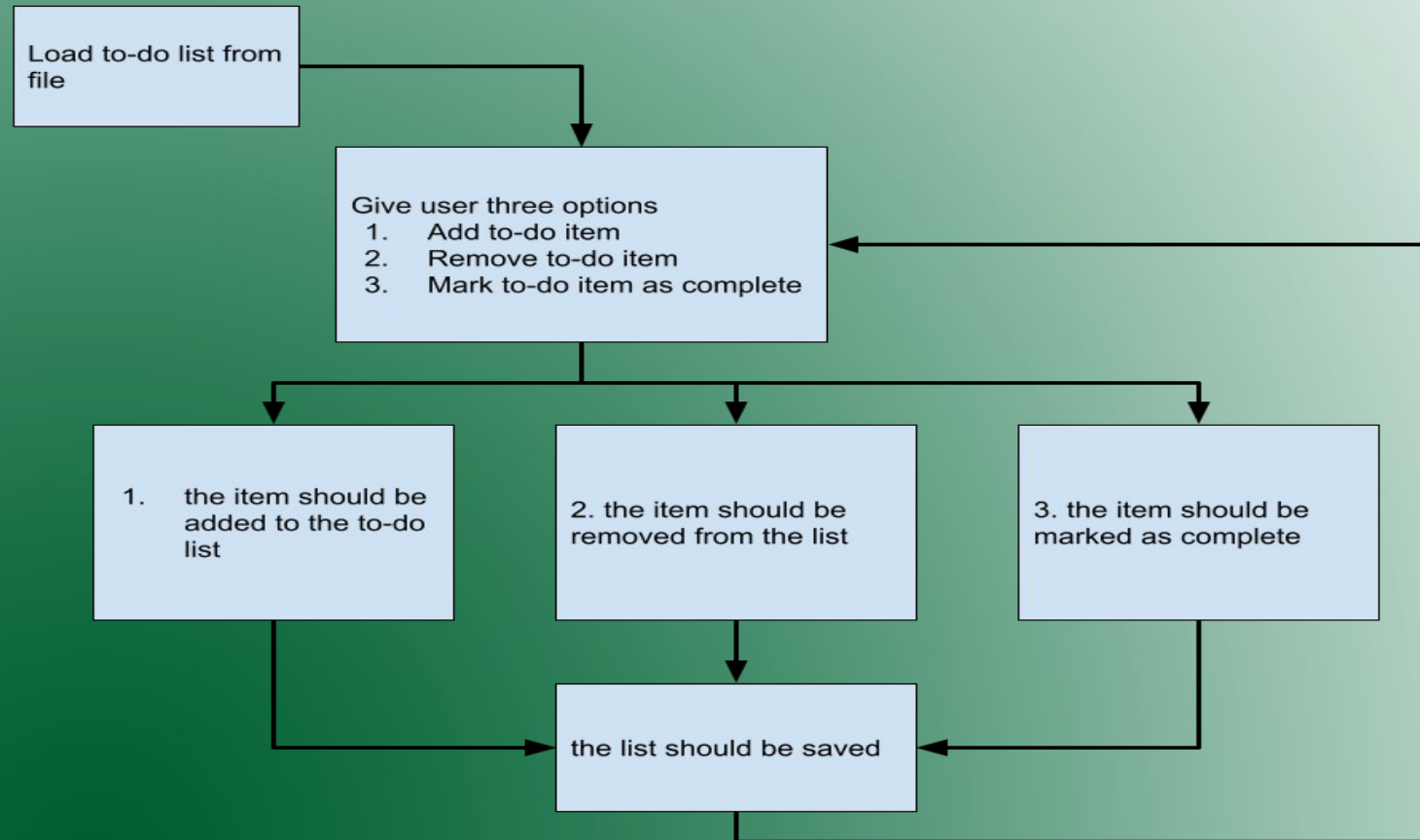
To-Do List Project

- How should a to-do list app work?
- What classes will we need to make?
- What variables will the classes need?
- What functions will the classes need?



The image shows a template for a To-Do List. It has a decorative border with a black and white polka-dot pattern. At the top, there is a white rectangular area containing the text "To-Do List" and a line for "week: ____". Below this, there is a list of 15 horizontal lines, each preceded by a small circle, representing a checklist.

How it should work



Open a file

- To open a file, we use the built-in function `open()`
- `open()` takes two arguments
 - *file*
 - *mode*

Open arguments

- `open(file, mode)`
 - file is the path and name of the file
 - **For example** *C:\Users\David Hunter\Documents\Seitoku Python Curriculum*
 - This is the path where I save my files for this class.
 - *C:\Users\David Hunter\Documents\Seitoku Python Curriculum/***2021 Python Class 10.odp**
 - The **BOLD** part is the file name.

Open arguments, 2

- `open(file, mode)`
 - mode is how you want to open the file
 - “r” -reads the file. Error if no file
 - “a” -appends. Creates the file if no file.
 - “w” - opens the file to write. Creates the file if no file
 - “x” - create the file. Error if the file exists.

Mode (ファイルの読み方)

- You can also say how the file should be read.
 - “b” is for binary (二進法) → People cannot read this format, but computers can.
 - “t” is for text

File variables

- file.closed – True or False
- file.mode- the mode
- file.name – the name
- ~~file.softspace – not really important~~

File functions

- `file.close()` - closes the file
- `file.read()` - this gives you the data/information in the file
- `file.write(str)` – this adds a string to the file
- `file.writelines(sequence)` – this adds a list of strings to the file

Practice

```
my_file = open("Hello.txt", "w")  
print(my_file.name)  
print(my_file.mode)  
#my_file.write("Hello")  
my_file.close()  
my_file = open("Hello.txt", "r")  
print(my_file.read())
```

Practice adding some information to a file

```
my_file = open("Hello.txt", "w")  
print(my_file.name)  
print(my_file.mode)  
my_file.write("Hello")  
my_file.close()  
my_file = open("Hello.txt", "r")  
print(my_file.read())
```

Practice reading information from a file

```
my_file = open("Hello.txt", "w")  
print(my_file.name)  
print(my_file.mode)  
my_file.write("Hello")  
my_file.close()  
my_file = open("Hello.txt", "r")  
data = my_file.read()
```

Using pickle to save data

- Pickle is a library, just like datetime or random.
- Pickle is used for saving data.
- It is especially useful for saving complicated data, such as class instances, lists, etc.

How to use

- Pickle has two main functions we will use
 - `pickle.dump(obj,file)`
 - `obj` is the information we want to save
 - `file` is a file that we opened
 - `pickle.load(file)`
 - this reads the data from the file and returns the data

Example

```
import pickle
import datetime

class Book:
    def __init__(self, n= "", a= "", d= datetime.date.today()):
        self.name = n
        self.author = a
        self.publish_date = d

    def __str__(self):
        return self.name + " by " + self.author + \
            " (" + str(self.publish_date.year) + ")"

book = Book("The Malazan Book of the Fallen", \
            "Steven Erikson", datetime.date(2001,1,1))

file = open("test","wb")
pickle.dump(book,file)

file.close()

infile = open("test","rb")

correct = pickle.load(infile)
infile.close()

print(correct)
```


Practice!!!

- Pick one of the projects we made in previous classes: DNA, recipe, manga-ka, vending machine, to-do list.
- Use pickle to save the data to a file.
- Use pickle to load the data.



DNA and RNA

- Why DNA/RNA?
 - All living things have DNA.
 - Even viruses, like covid-19, use DNA or RNA to store their genetic information(遺伝子情報).

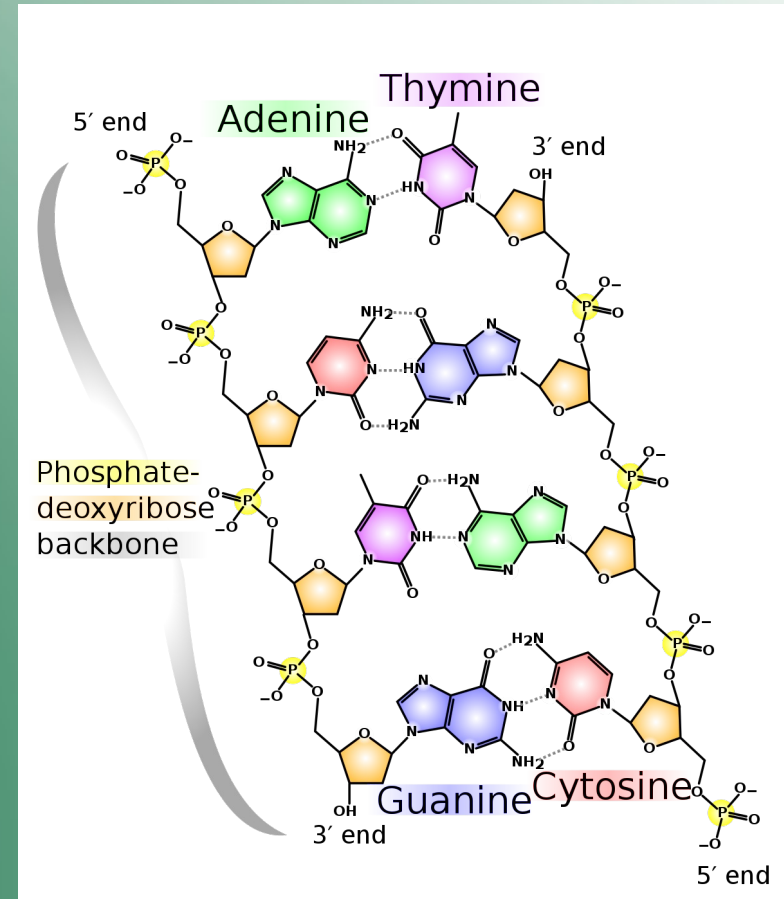
The Goal

- Make a DNA class with a list of genes
- Make 3 different DNA instances
 - Influenza has 8 genes
 - MERS has 11 genes
 - Covid-19 has 9 genes
- Let's compare them!

DNA Information

- DNA controls genes (遺伝子).
- Genes create different proteins (酵素 / タンパク質).

DNA Information 2

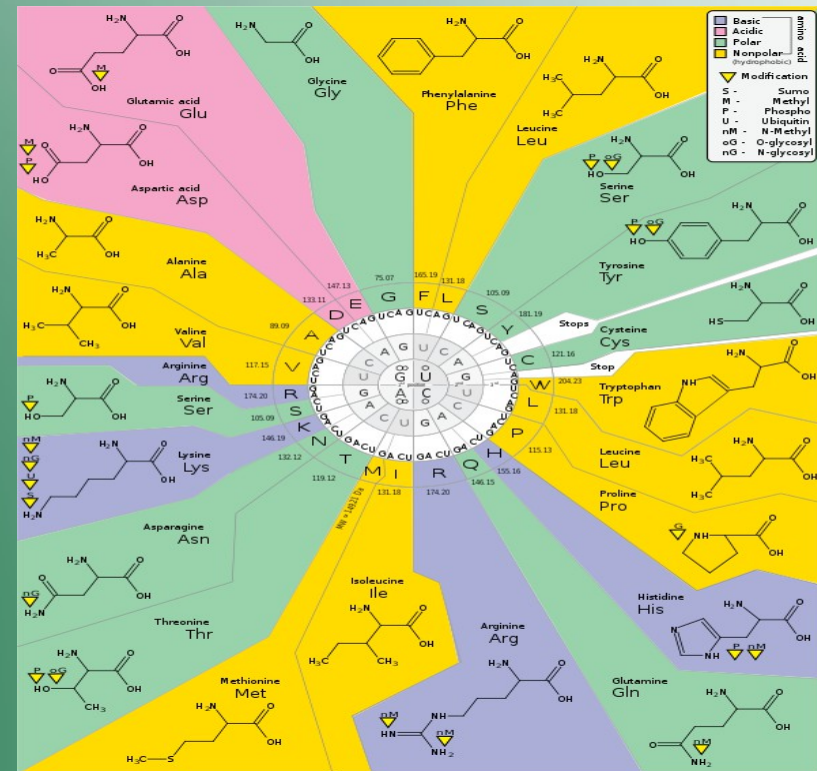
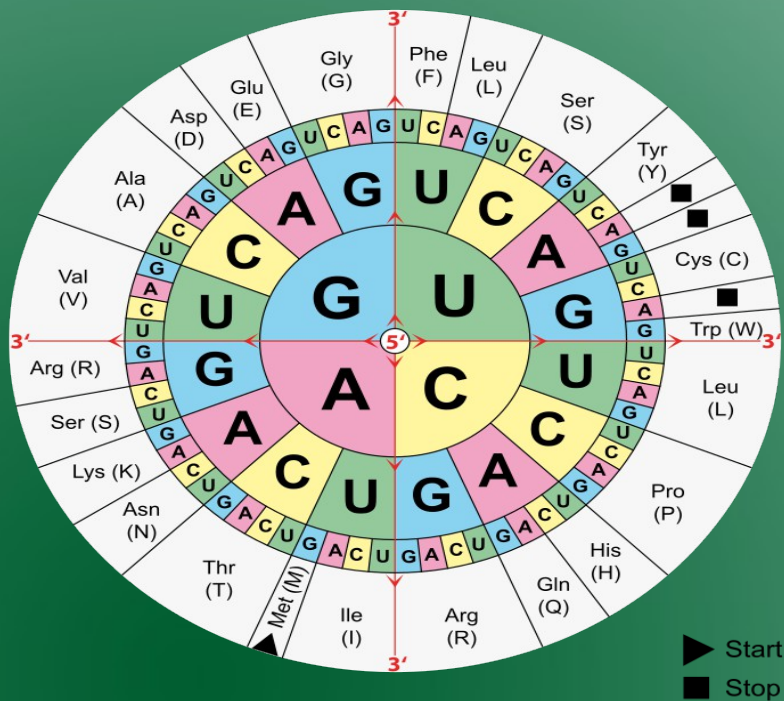


DNA Information 3

- DNA has different chemicals in groups of three.
- These chemicals are
 - Adenine
 - Thymine
 - Guanine
 - Cytosine

DNA, 4

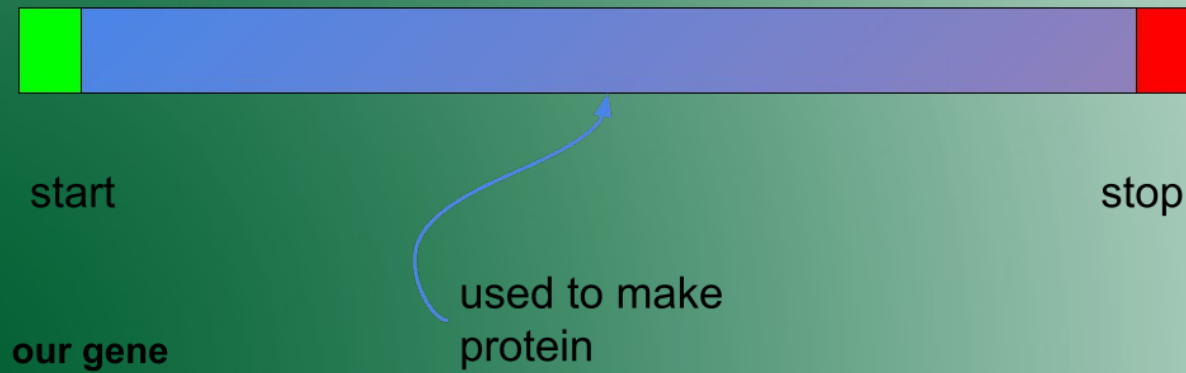
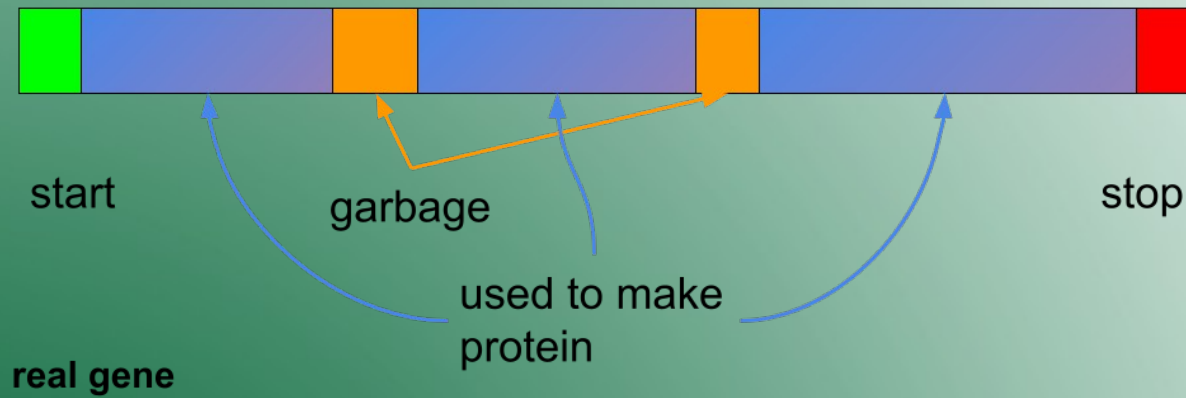
- Each group of three chemicals creates a part of a protein.



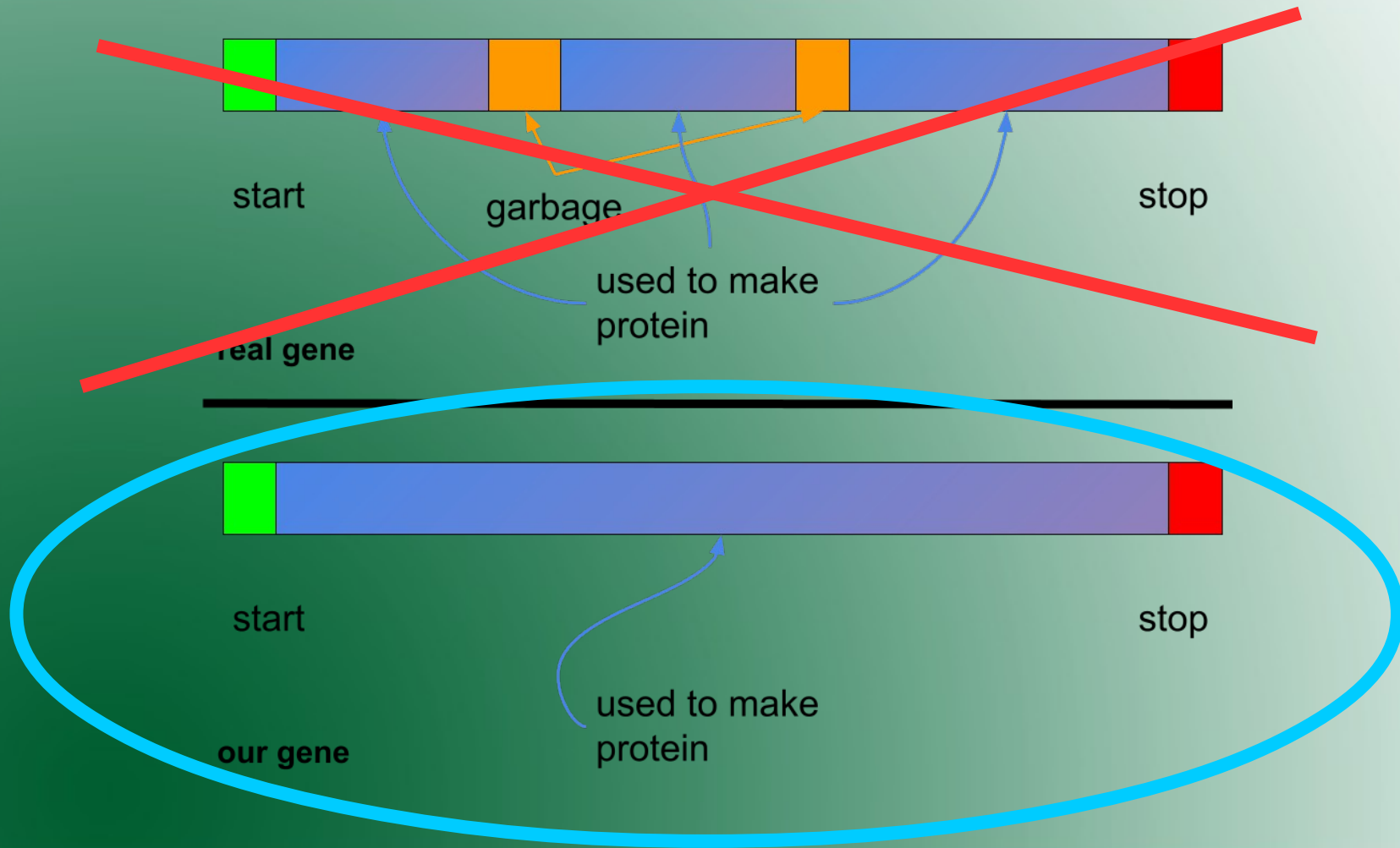
DNA 5

- Every gene starts with a special group of three
 - ATG (Adenine Thymine Guanine)
- There are three ways to stop a gene
 - TAG (Thymine Adenine Guanine)
 - TGA (Thymine Guanine Adenine)
 - TAA (Thymine Adenine Adenine)

Gene Structure



Gene Structure



DNA exercise as list

- Let's write some code to make a random gene!

```
import random
```

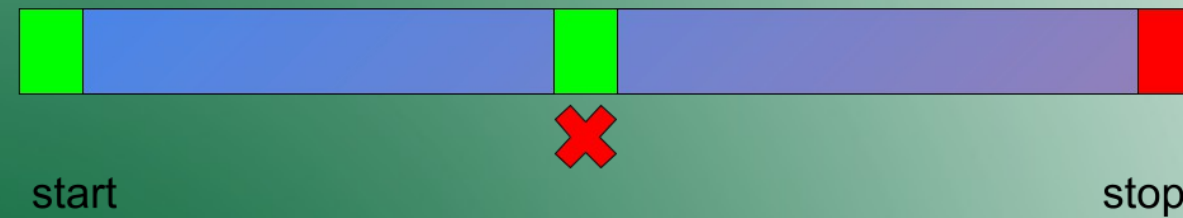
```
#we will use random.randint(a,b)
```

```
chemicals = ["A", "T", "C", "G"]
```

```
start = "ATG"
```

```
end = ["TAG", "TGA", "TAA"]
```

Rule for making codons



Let's change our code to a gene class



- What variables will it need?
- What functions will our gene class need?