Welcome to the Labs!

Secret Diary Chatbot!

Thank you to our Sponsors!

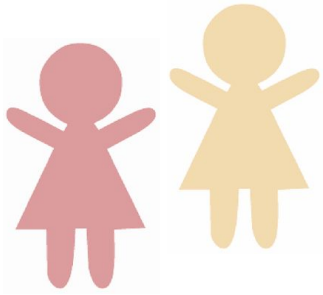
Platinum Sponsor:

Who are the tutors?

Who are you?

Two Truths and a Lie

1. Get in a group of 3-5 people
2. Tell them three things about yourself:
   a. Two of these things should be true
   b. One of these things should be a lie!
3. The other group members have to guess which is the lie



Log on

Log on and jump on the GPN website

girlsprogramming.network/workshop

You can see:

- These **slides** (to take a look back or go on ahead).
- A digital copy of your **workbook**.
- Help bits of text you can **copy and paste**!

There's also links to places where you can do more programming!

Tell us you're here!

Click on the
Start of Day Survey
and fill it in now!

Introduction to Edstem

Signing up to Edstem

We are shifting all our courses to a new website called “Edstem” so here’s an overview of how to sign up and how to use it.

First let’s go through how to create an account.

1. Follow this join link: https://edstem.org/au/join/4KxYP6
2. Put in your name and your personal email address
3. Click Create Account
4. Go to your email to verify your account
5. Create a password
6. It should then take you to the courses home page. Click on the one we will be using for this project; Chatbot N

If you don’t have access to your email account, ask a tutor for a GPN edStem login

Getting to the lessons

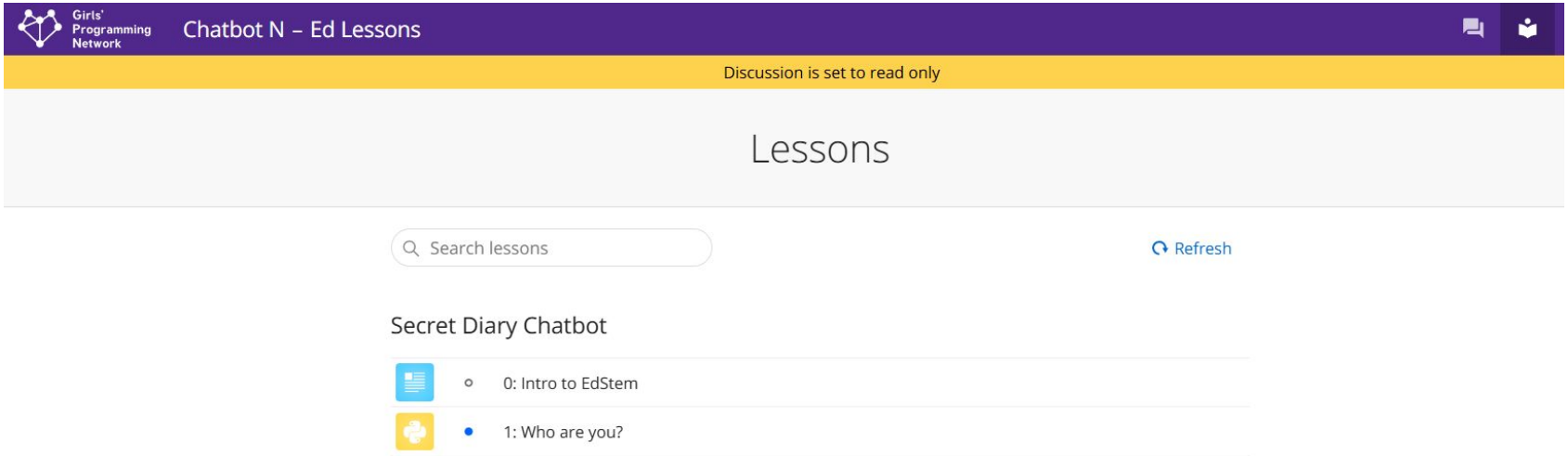
Once you are in the course, you'll be taken to a discussion page.

Click the button for the lessons page (top right - looks like a book)

The Anatomy of the workbook

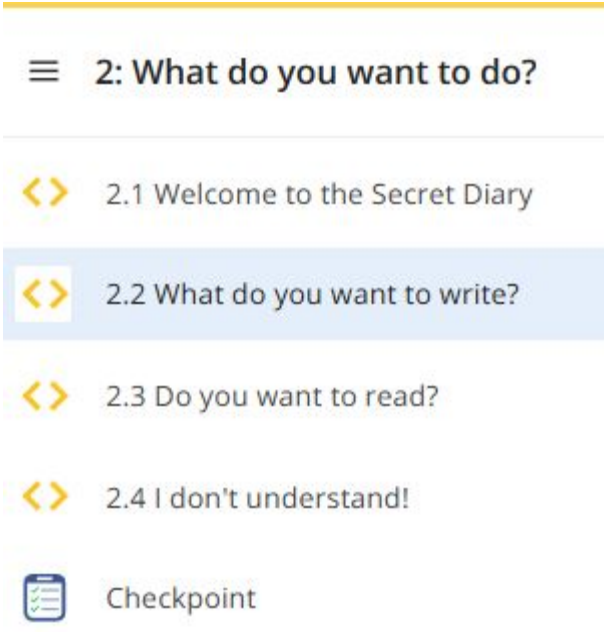
The main page:

- Heading at the top that tells you the project (ChatbotN)
- List of “Chapters” - they have icons that looks like this:
- To complete your project, work through the chapters one at a time



Inside a Chapter

Inside a chapter there are two main types of pages:



1. **Lessons** - where you will do your coding.

   They have this icon: 

2. **Checkpoints**  Checkpoint

   Each chapter has a checkpoint to complete to move to the next chapter. Make sure you scroll down to see all the questions in a checkpoint.

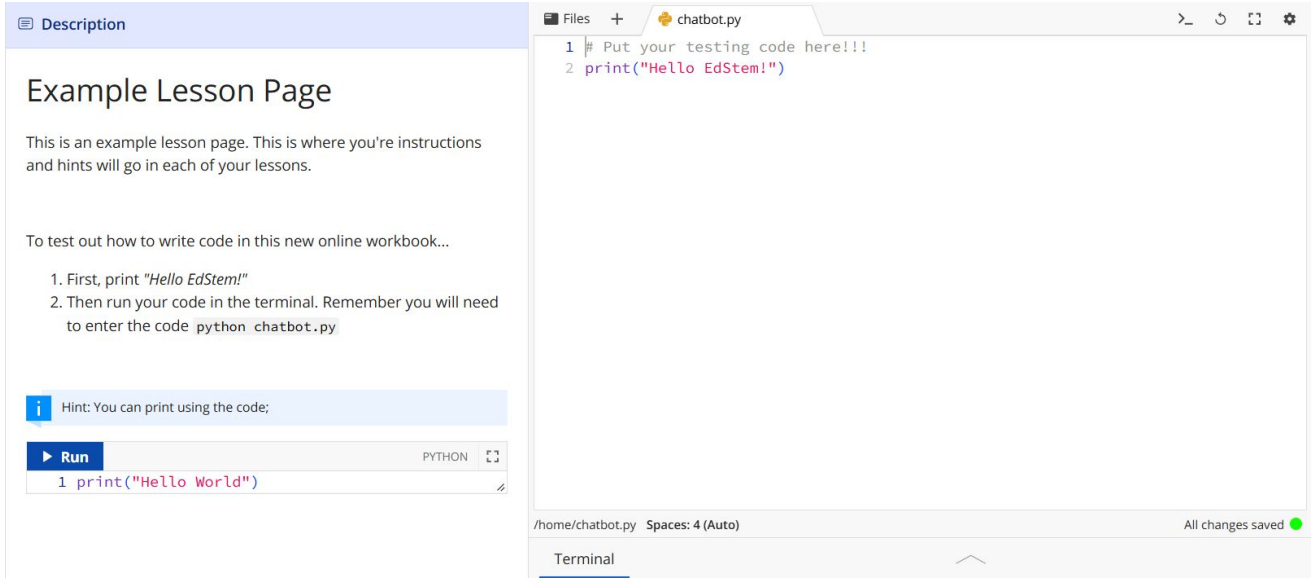
How to do the work

In each lesson there is:

- A section on left with instructions
- A section on right for your code

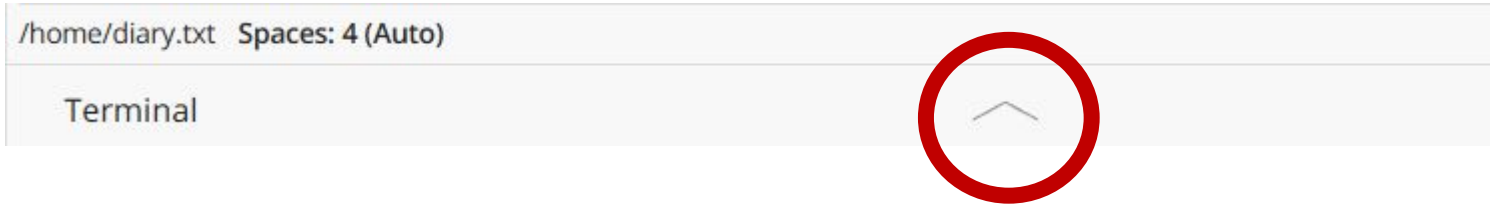
You will need to **copy your code from the last lesson**, then follow the instructions to change your code

There are also Hints and Code Blocks to help you



Running your code...

1. Open the Terminal window below your code



2. Click button that says “Click here to activate the terminal”.

3. Your code should run automatically.
4. Click the button again to rerun your code.
5. You can resize the Terminal window.

Don’t worry if you forget. Tutors will help!

Some shortcuts...

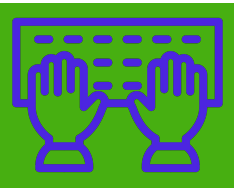
There are a couple things you can do to make copying your code from one page to another easier.

1) **Ctrl + A** Pressing these keys together will select all the text on a page

2) **Ctrl + C** Pressing these keys together will copy anything that's selected

3) **Ctrl + V** Pressing these keys together will paste anything you've copied

Need help with EdStem?

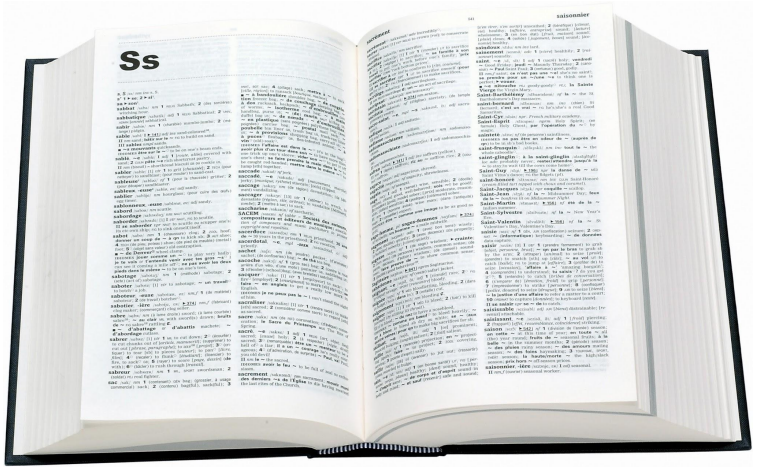


There is a section at the top of your workbook that explains how to use EdStem if you get stuck and need a reminder!

It's called 0: Intro to EdStem

Go to Part 0 and have a look!

Files, Dictionaries, & Functions



Opening files!

To get access to the stuff inside a file in python we need to **open** it!

That doesn't mean clicking on the little icon!

```
f = open("test.txt")
```

You'll now be able to read the things in `f`

If your file is in the same location as your code you can just use the name!

A missing file causes an error

Here we try to open a file that doesn't exist:

```
f = open('missing.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or
directory: 'missing.txt'
```

You can read a whole file into a string

```
>>> f = open('haiku.txt')
>>> my_string = f.read()
```

haiku.txt

```
Wanna go outside.
Oh NO! Help! I got outside!
Let me back inside!
```

```
>>> print(my_stirng)
Wanna go outside.
Oh NO! Help! I got outside!
Let me back inside!
```

You can also read in one line at a time

You can use a for loop to only get 1 line at a time!

```
f = open('haiku.txt')
for line in f:
    print(line)
```

```
Wanna go outside.

Oh NO! Help! I got outside!

Let me back inside!
```

Why is there an extra blank line each time?

Chomping off the newline

The newline character is represented by '\n':

```
print('Hello\nWorld')
Hello
World
```

We can remove it from the lines we read with .strip()

```
x = 'abc\n'
x.strip()
'abc'
```

x.strip() is safe as lines without newlines will be unaffected

Reading and stripping!

```
for line in open('haiku.txt'):
    line = line.strip()
    print(line)
```

```
Wanna go outside.
Oh NO! Help! I got outside!
Let me back inside!
```

No extra lines!

Using with!

This is a special trick for opening files!

```
with open("words.txt") as f:
  for line in f:
    print(line.strip())
```

It automatically closes your file for you!

It's good when you are writing files in python!

Write to files!

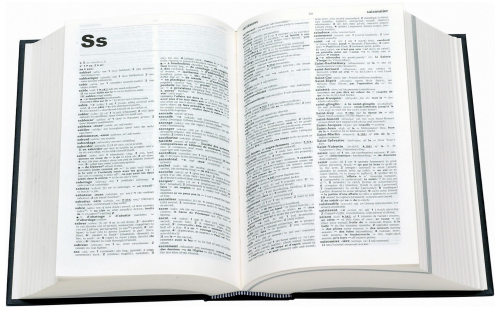
You can also write to files!

```
f = open("newfile.txt", "w")
f.write("This is my new text!")
```

Notice we used `"w"` instead of `"r"`? We opened it in write mode!

This will create a new file if it doesn't exist, and overwrite the file if it already exists

Dictionaries!

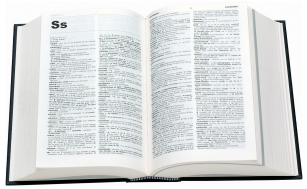


You know dictionaries!

They're great at looking up a thing by a word, not a position in a list!

Look up

Hello



Get back

A greeting (salutation) said when meeting someone or acknowledging someone's arrival or presence.

Looking it up!

There are lots of times we want to look something up!

Competition registration

Team Name → List of team members

Phone Book

Name → Phone number

Vending Machine

Treat Name → Price

Looking it up!

Phone Book

Name → Phone number

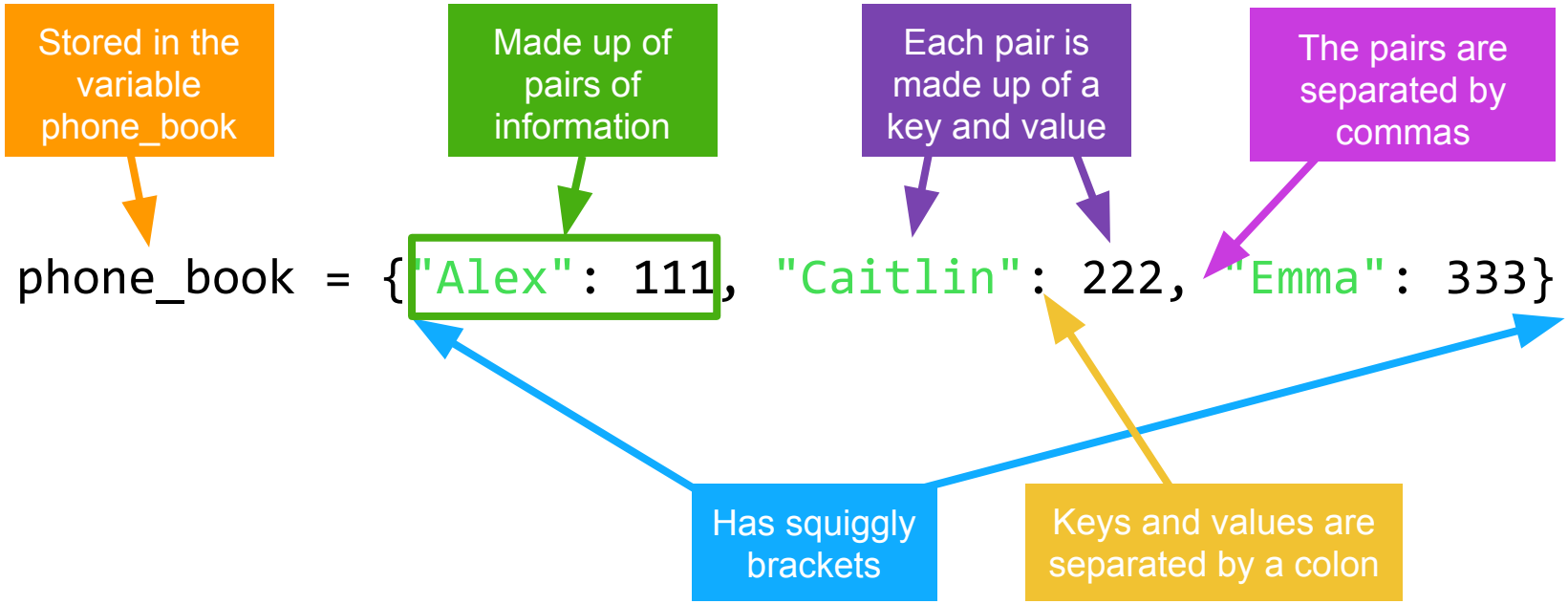
Key

Value

We can use a dictionary for anything with a <u>key → value</u> pattern!

Dictionaries anatomy!

This is a python dictionary!



This dictionary has Alex, Caitlin and Emma's phone numbers

Playing with dictionaries!

Let's look at an example!

```
phone_book = {"Alex": 111, "Caitlin": 222, "Emma": 333}
```

1. **What happens?**

   ```
   phone_book["Alex"]
   ```

2. **How would you look up Emma's phone number?**

3. **Look up the name of someone who is not in the phone book? What happens?**

Playing with dictionaries!

Let's look at an example!

```
phone_book = {"Alex": 111, "Caitlin": 222, "Emma": 333}
```

1. **What happens?**

   `phone_book["Alex"]` 111

2. **How would you look up Emma's phone number?**

3. **Look up the name of someone who is not in the phone book? What happens?**

Playing with dictionaries!

Let's look at an example!

```
phone_book = {"Alex": 111, "Caitlin": 222, "Emma": 333}
```

1. **What happens?**

   `phone_book["Alex"]` 111

2. **How would you look up Emma's phone number?**

   `phone_book["Emma"]`

3. **Look up the name of someone who is not in the phone book? What happens?**

Playing with dictionaries!

Let's look at an example!

```
phone_book = {"Alex": 111, "Caitlin": 222, "Emma": 333}
```

1. **What happens?**

   `phone_book["Alex"]` 111

2. **How would you look up Emma's phone number?**

   `phone_book["Emma"]`

3. **Look up the name of someone who is not in the phone book? What happens?**

   `KeyError`

Is it in the dictionary?

What if we want to check whether a name is already in our contacts so we don't add it again? for that we'll need to check if the name is `in` the dictionary.**Take this example...**

```
contacts = {"John": 11, "Jane": 22, "Jack": 33}
```

What happens if we say;

```
>>> if "Jane" in contacts:
...     print("Hi Jane!")
... else:
...     print("I don't know you :(")
```

Is it in the dictionary?

What if we want to check whether a name is already in our contacts so we don't add it again? for that we'll need to check if the name is `in` the dictionary.**Take this example...**

```
contacts = {"John": 11, "Jane": 22, "Jack": 33}
```

What happens if we say;

```
>>> if "Jane" in contacts:
...     print("Hi Jane!")
... else:
...     print("I don't know you :(")
Hi Jane!
```

Is it in the dictionary?

What if we want to check whether a name is already in our contacts so we don't add it again? for that we'll need to check if the name is `in` the dictionary.**Take this example...**

```
contacts = {"John": 11, "Jane": 22, "Jack": 33}
```

But what happens if we change the name?

```
>>> if "Renee" in contacts:
...     print("Hi Renee!")
... else:
...     print("I don't know you :(")
```

Is it in the dictionary?

What if we want to check whether a name is already in our contacts so we don't add it again? for that we'll need to check if the name is `in` the dictionary.**Take this example...**

```
contacts = {"John": 11, "Jane": 22, "Jack": 33}
```

But what happens if we change the name?

```
>>> if "Renee" in contacts:
...     print("Hi Renee!")
... else:
...     print("I don't know you :(")
I don't know you :(
```

Looping through a dictionary

Now what if we want to look at every element in the dictionary one by one?

Looping through a dictionary

Now what if we want to look at every element in the dictionary one by one?

For loops!

Looping through a dictionary

Now what if we want to look at every element in the dictionary one by one?

For loops!

Using for loops we can go through each key of the dictionary one by one.

Looping through a dictionary

```
contacts = {“John”: 11, “Jane”: 22, ”Jack”: 33}
for i in contacts:
    print(i)
```

What’s going to happen?

Looping through a dictionary

```
contacts = {“John”: 11, “Jane”: 22, ”Jack”: 33}
for i in contacts:
    print(i)
```

What’s going to happen?

```
>>> John
>>> Jane
>>> Jack
```

- Each key in the dictionary takes a turn at being the variable i
- Do the body once for each item
- We’re done when we run out of items!

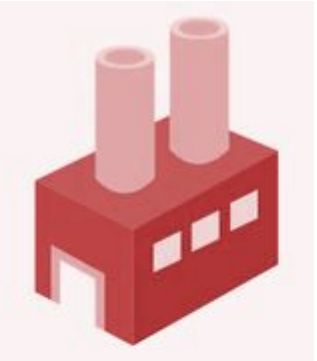
How functions fit together!

Functions are like factories!

Timber Mill

Your main factory!

Metal Worker

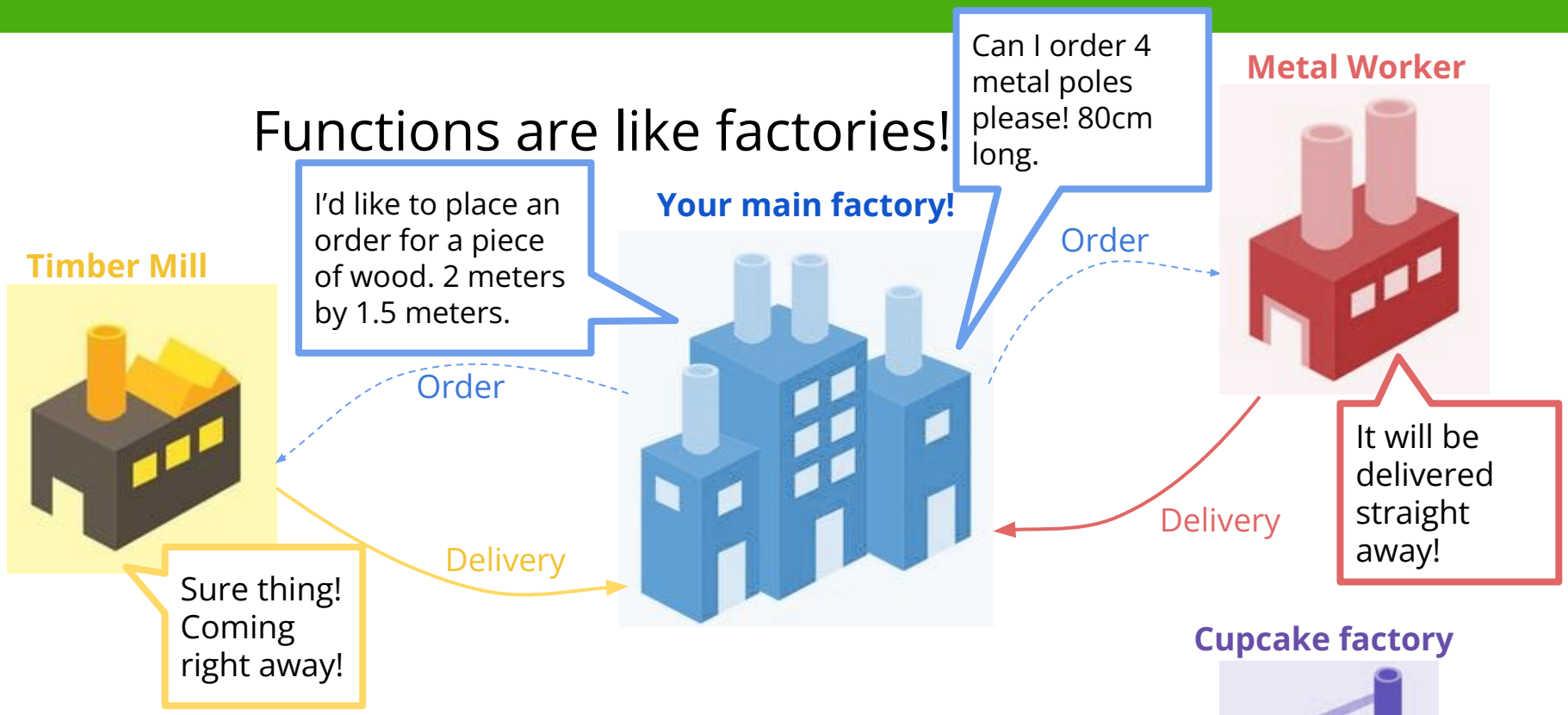


Cupcake factory

Running a factory doesn't mean doing all the work yourself, you can get other factories to help you out!

How functions fit together!

Functions are like factories!



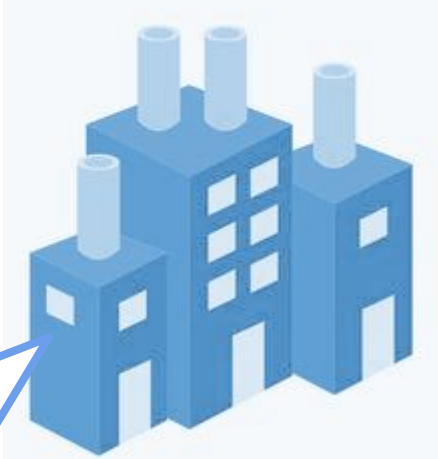
Asking other factories to do some work for you makes your main task simper. You can focus on the assembly!

How functions fit together!

Functions are like factories!

Metal Worker

Your main factory!



Timber Mill

Look at this beautiful table I made!

Outsourcing made it simple!

Cupcake factory

How functions fit together!

Your main code!

You can write a bunch of helpful functions to **simplify** your **main goal**!

You can **write** these **once** and then **use** them **lots** of times! They can be **anything** you like!

Helps with printing nicely

Uses stats to make decisions

Does calculations

Don’t reinvent the wheel

We’re already familiar with some python in built functions like print and input!

There’s lots of functions python gives us to save us reinventing the wheel!

For instance we can use len to get the length of a string, rather than having to write code to count every letter!

```
>>> len("Hello world")
11
```

Try these:

```
>>> name = "Renee"
>>> len(name)
5

>>> int("6")
6

>>> str(6)
"6"
```

Defining your own functions

Built in functions are great! But sometimes we want custom functions!

Defining our own functions means:

- We cut down on repeated code
- Nice function names makes our code clear and easy to read
- We can move bulky code out of the way

Defining your own functions

Then you can use your function by calling it!

```
def cat_print():
    print("""
                        #
                         #
                         #
                ^..^ #####
                =TT=      ;
                 #########
                 # #   # #
                 M M   M M """)


cat_print()
cat_print()
```

Which will do this!

```
                        #
                         #
                         #
                ^..^ #####
                =TT=      ;
                 #########
                 # #   # #
                 M M   M M
                        #
                         #
                         #
                ^..^ #####
                =TT=      ;
                 #########
                 # #   # #
                 M M   M M
```

Defining your own functions

Then you can use your function by calling it!

```
def cat_print():
    print("""
                        #
                         #
                         #
                ^..^ #####
                =TT=      ;
                 #########
                 # #   # #
                 M M   M M
                        #
                         #
                         #
                ^..^ #####
                =TT=      ;
                 #########
                 # #   # #
                 M M   M M
cat_print()
cat_print()
```

When using a function in a **script** make sure you define the function first.

It doesn't matter if you call it from inside another function though!

Which will do this!

```
                        #
                         #
                         #
                ^..^ #####
                =TT=      ;
                 #########
                 # #   # #
                 M M   M M
                        #
                         #
                         #
                ^..^ #####
                =TT=      ;
                 #########
                 # #   # #
                 M M   M M
```

Functions often need extra information

Functions are more useful if we can change what they do

We can do this by giving them arguments (aka parameters)

```
>>> def hello(person):
...     print('Hello, ' + person + ', how are you?')
>>> hello('Alex')
Hello, Alex, how are you?
```

Here, we give the hello() function a name

Any string will work

```
>>> hello('abcd')
Hello, abcd, how are you?
```

Functions can take multiple arguments

Often we want to work with multiple pieces of information.

You can actually have as many parameters as you like!

This function takes two numbers, adds them together and prints the result.

```
>>> def add(x, y):
...     print(x + y)
>>> add(3, 4)
7
```

Arguments stay inside the function

The arguments are not able to be accessed outside of the function declaration.

```
>>> def hello(person):
...     print('Hello, ' + person + '!')
>>> print(person)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'person' is not defined
```

Variables stay inside the function

Neither are variables made inside the function. They are **local variables**.

```
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
>>> z
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'z' is not defined
```

Global variables are not affected

Changing a variable in a function **only changes it *inside* the function.**

```
>>> z = 1
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

Global variables are not affected

Changing a variable in a function **only changes it *inside* the function.**

```
>>> z = 1
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

What's the value of z now?

```
>>> print(z)
```

Global variables are not affected

Changing a variable in a function **only changes it *inside* the function.**

```
>>> z = 1
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

What's the value of z now?

```
>>> print(z)
1
```

More on global variables

But what if we want to be able to change a variable that is made outside a function **INSIDE** the function?

For that we need to declare the variable as `global`

Let's take our old code and see how it will change if we make z global...

More on global variables

But what if we want to be able to change a variable that is made outside a function **INSIDE** the function?

For that we need to declare the variable as `global`

Let's take our old code and see how it will change if we make z global...

```
>>> z = 1
>>> def add(x, y):
...     global z
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

What should this print now?

```
>>> print(z)
```

More on global variables

But what if we want to be able to change a variable that is made outside a function **INSIDE** the function?

For that we need to declare the variable as `global`

Let's take our old code and see how it will change if we make z global...

```
>>> z = 1
>>> def add(x, y):
...     global z
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

What should this print now?

```
>>> print(z)
7
```

Giving something back

At the moment our function just does a thing, but it's not able to give anything back to the main program.

Currently, we can't use the result of add()

```
>>> def add(x, y):
...     print(x + y)
>>> sum = add(1, 3)
4
>>> sum
```

sum has no value!

Giving something back

Using `return` in a function immediately returns a result.

```
>>> def add(x, y):
...     z = x + y
...     return z
...
>>> sum = add(1, 3)
>>> sum
4
```

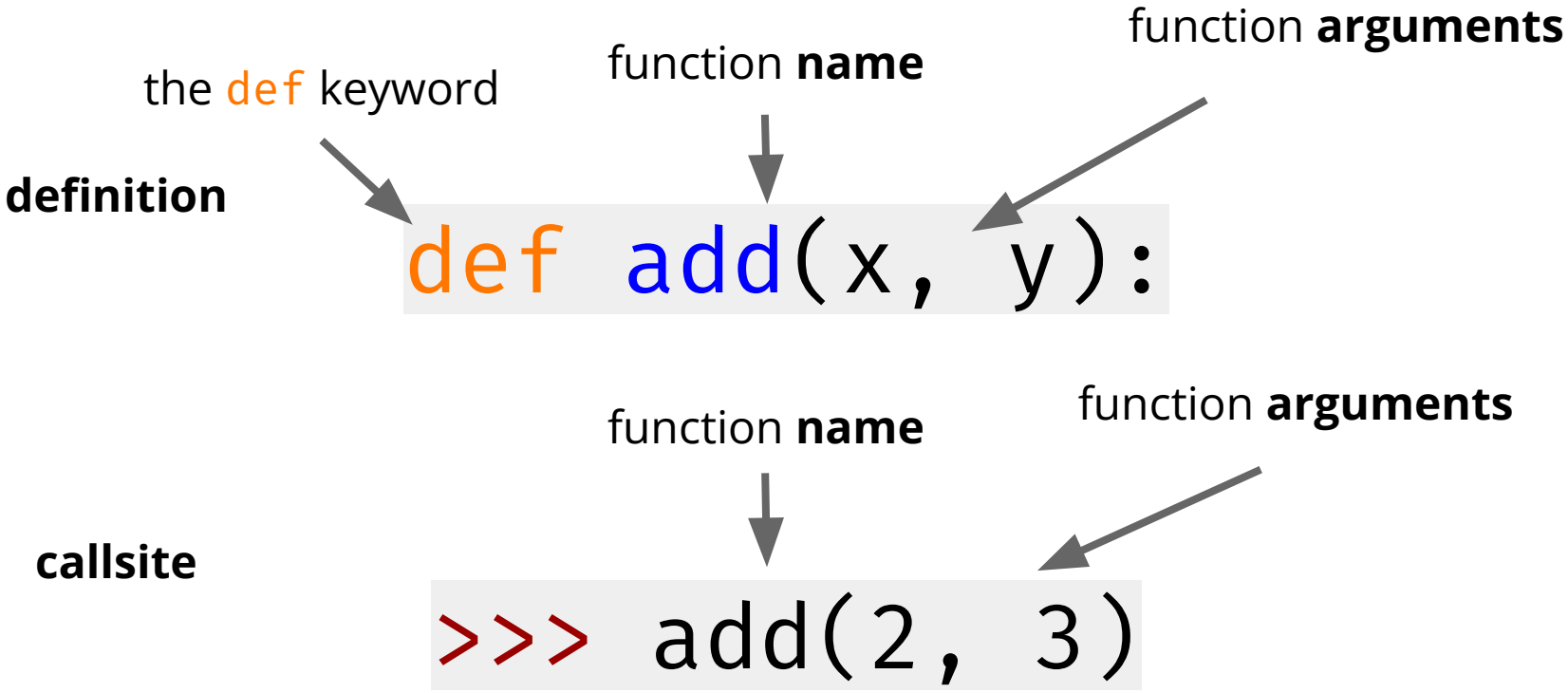
Giving something back

When a function returns something, the *control* is passed back to the main program, so no code after the `return` statement is run.

```
>>> def add(x, y):
...     print('before the return')
...     z = x + y
...     return z
...     print('after the return')
>>> sum = add(1, 3)
before the return
>>> sum
4
```

Here, the `print` statement after the `return` never gets run.

Recap: Function Anatomy



Project time!

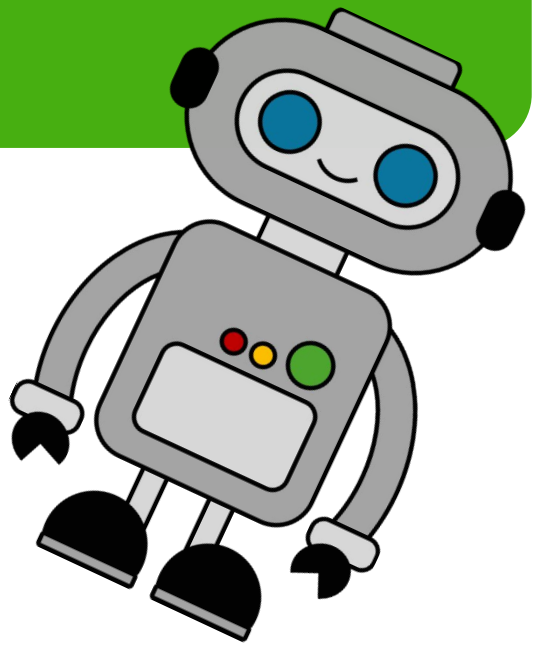
You now know all about files, dictionaries, and functions!

Let’s put what we learnt into our project

Try to do Part 1 - 5

The tutors will be around to help!

Regex



What is Regex?

Regex or “Regular Expression” is a way of searching text for parts that match a certain “pattern”

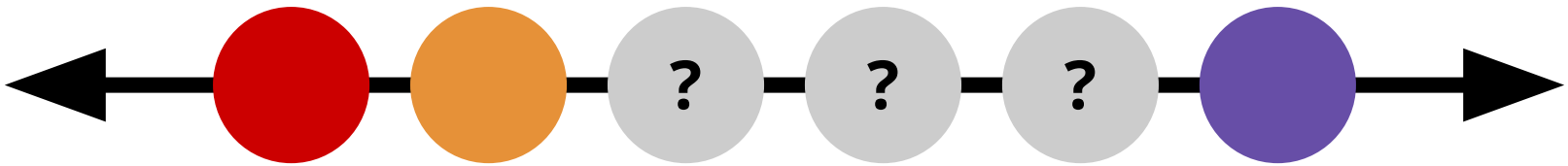
Like how an email will fit the pattern text@gmail.com

Or a phone number will fit the pattern (+area code) 8 numbers

How does this work?

Let's do a simulated version of this to see how it works.

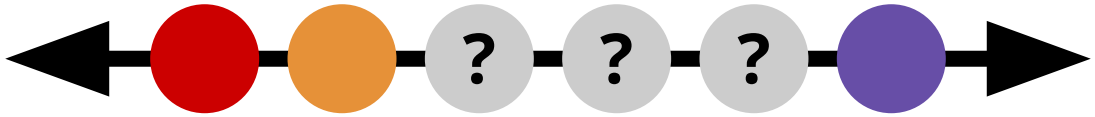
Let's say we want to find a pattern of beads that goes red, orange, 3 beads, then purple.

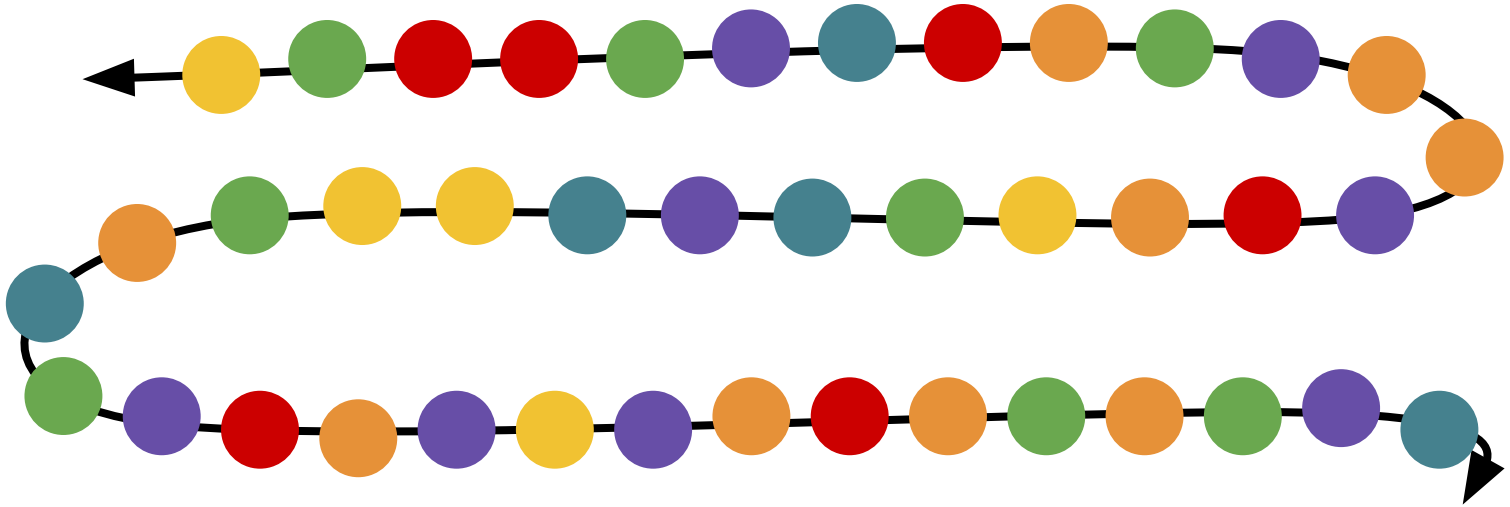


Let's work through how the computer would search a whole bracelet.

How does this work?

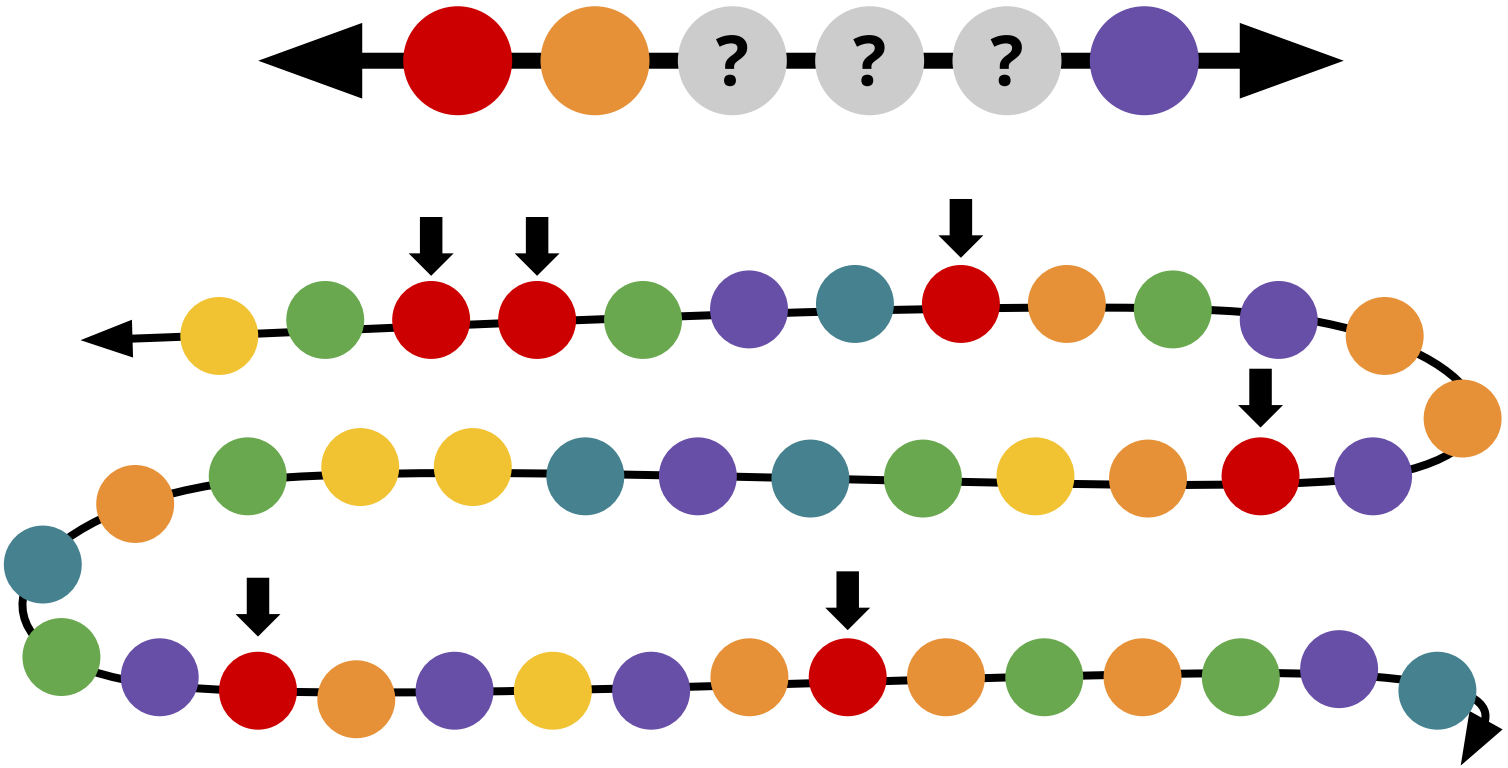
First the computer will go through the bracelet to see if any match the first element in the pattern (a red) bead





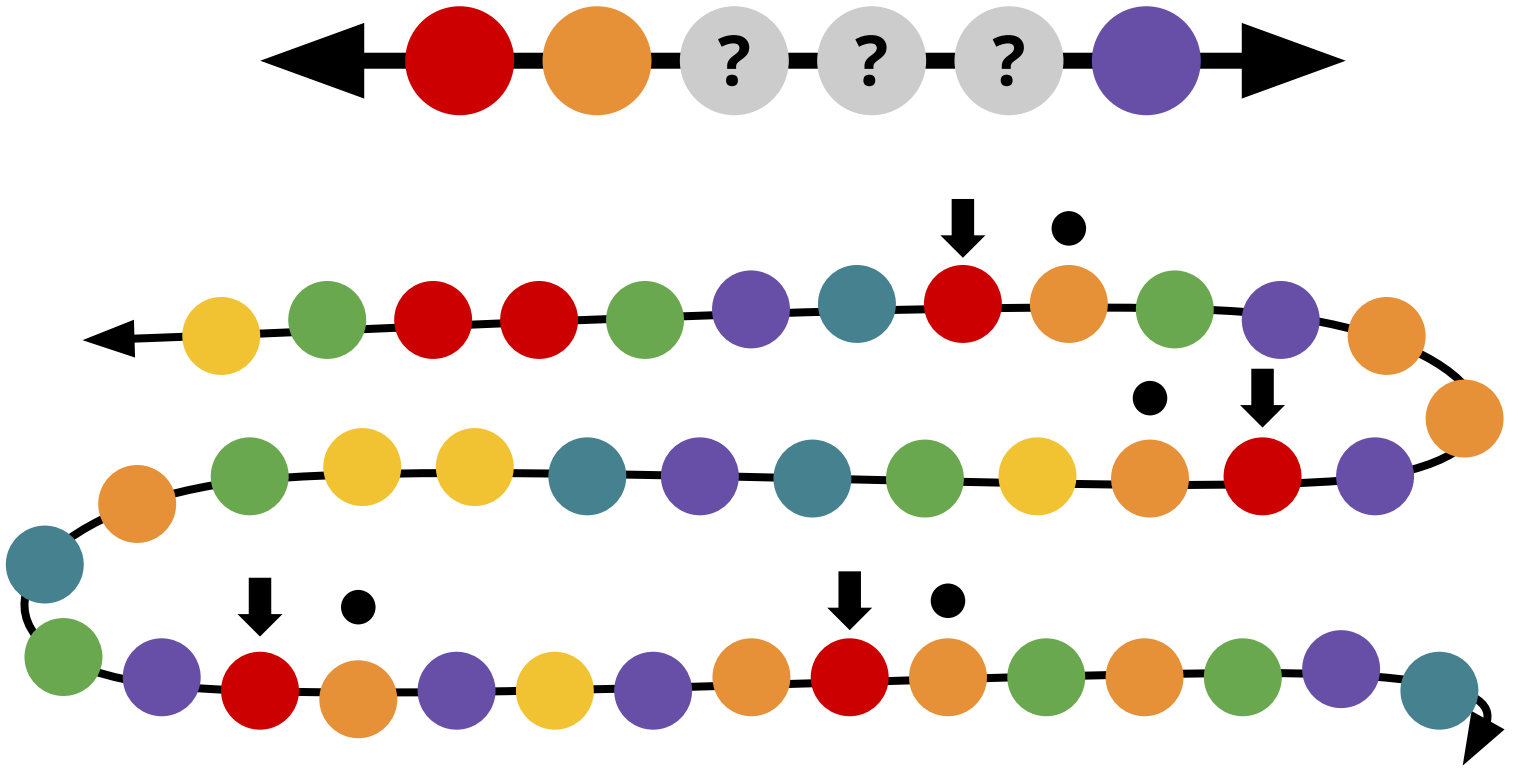
How does this work?

First the computer will go through the bracelet to see if any match the first element in the pattern (a red) bead



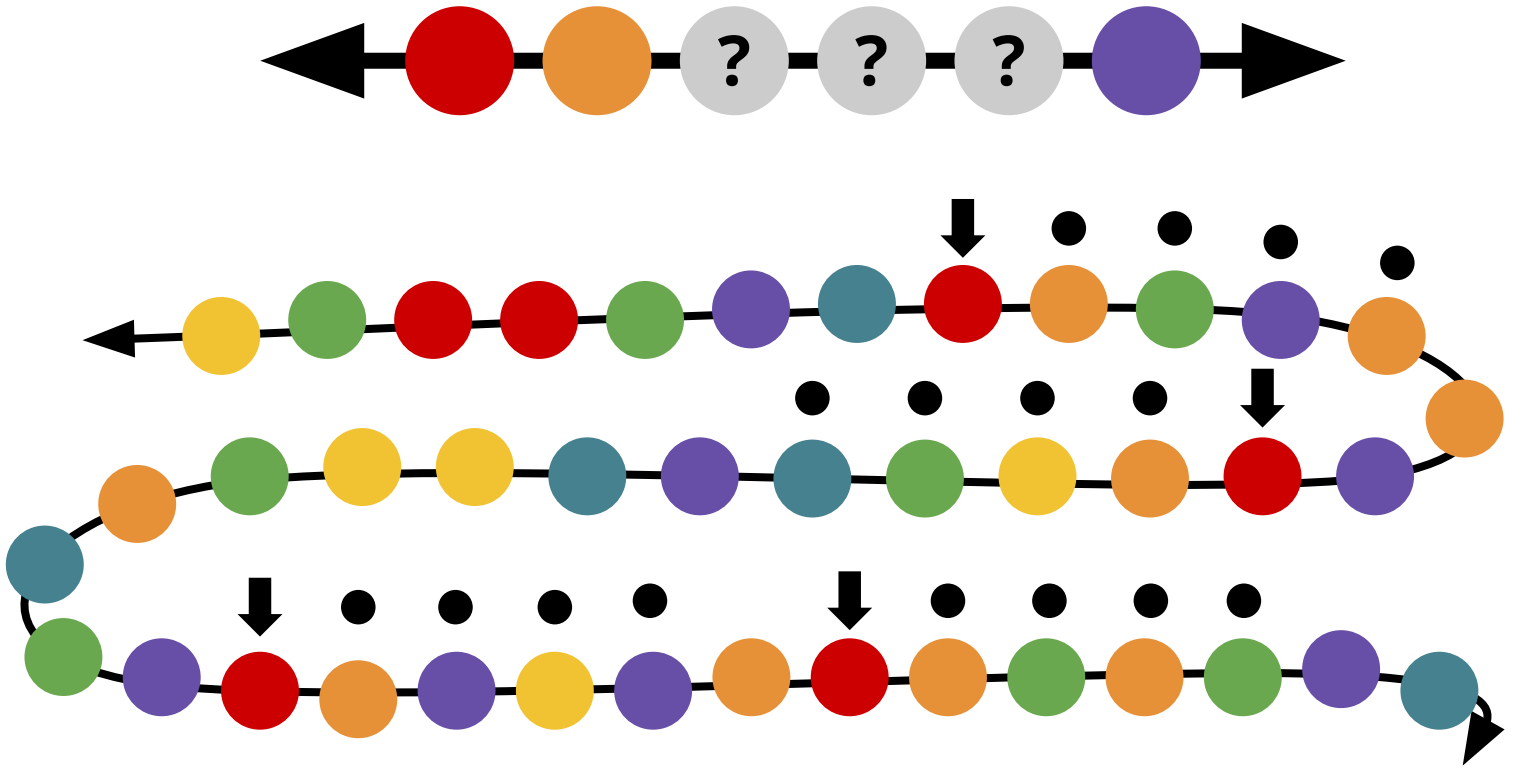
How does this work?

Then it will go through and check if the next bead along also matches (is orange), and rule out the ones that don't



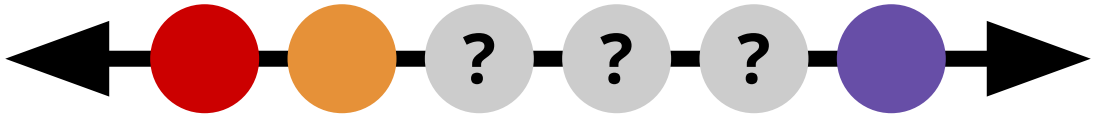
How does this work?

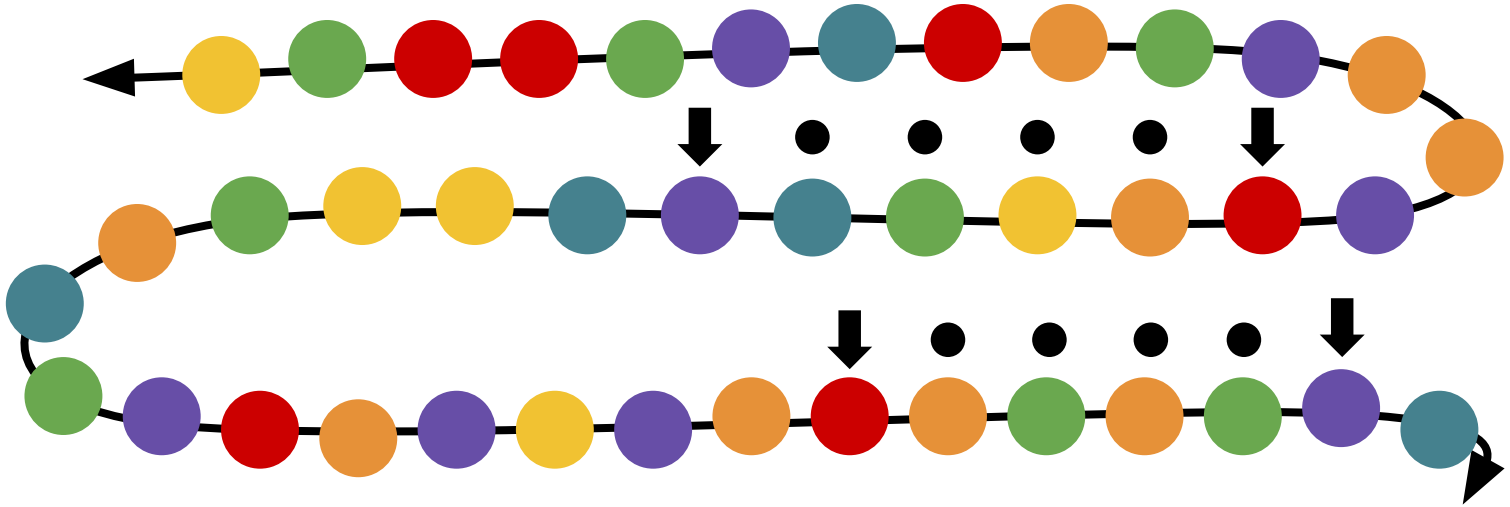
Next in the pattern it will accept any three beads, so it'll automatically go ahead three beads.



How does this work?

The pattern then says the last bead must be purple, so it goes through and checks if the last bead is purple and if it isn't it is ruled out

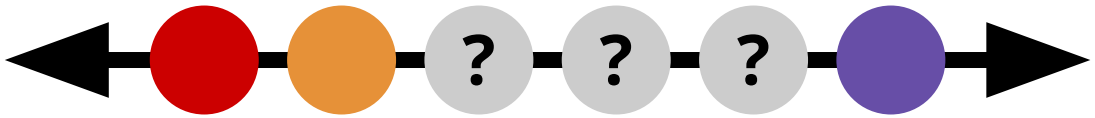




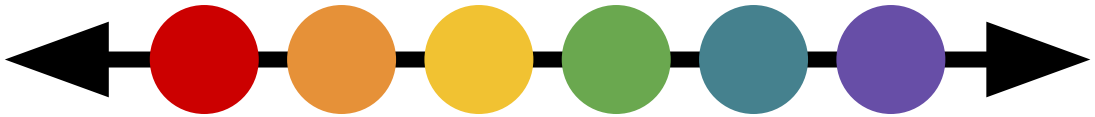
The result...

From searching our bracelet we have found two sections that match our pattern...

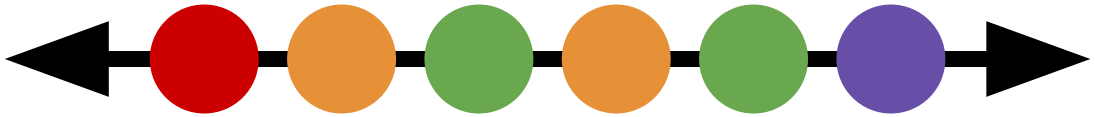
The pattern:



Match 1:



Match 2:



Beads to text

Let's write our bead pattern in regex, assuming that red is "r", orange is "o", and purple is "p".

"ro...p"

Here the main building blocks are:

- The letters (they mean that exact letter has to be present)
- And the full stops. They are a placeholder for any character.

How to make a regex “pattern”

The main idea of regex is to search a string for sections that fit a certain “pattern”. The main building blocks of a pattern are;

| means “or” e.g. **(a|b)** means a or b

\d represents any number e.g. **pass\d** will accept pass1, pass2 etc.

() groups items together

^ represents the beginning of a line

$ represents the end of a line

\b represents the start or end of a “word”

a{1,5} means there must be between 1 and 5 of the letter a

a+ represents one or more of the letter a

Some examples

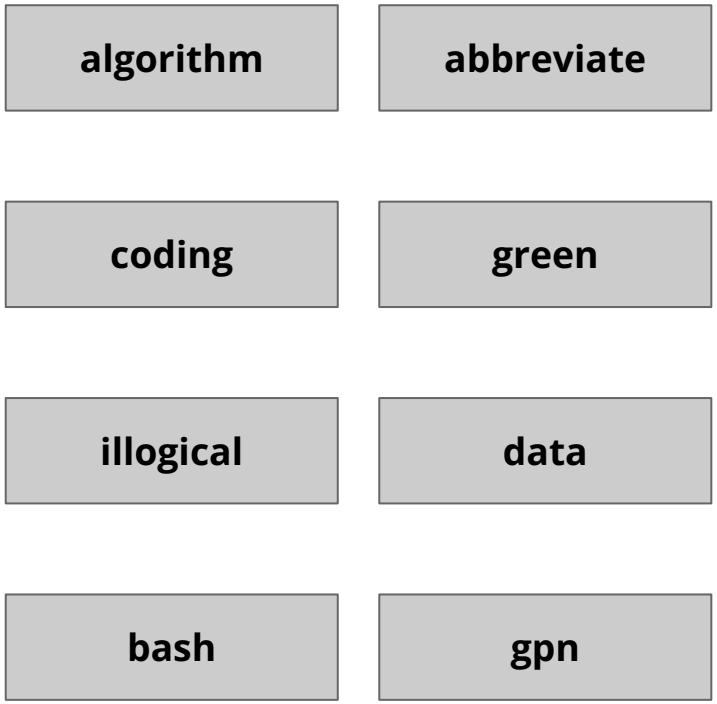
Match these patterns to the word examples

```
>>> d
```

```
>>> ^g
```

```
>>> ^.l+
```

```
>>> ^(b|a)+
```



Some examples

Match these patterns to the word examples

```
>>> d
```

data

coding

```
>>> ^g
```

```
>>> ^.l+
```

```
>>> ^(b|a)+
```

algorithm

abbreviate

green

illogical

bash

gpn

Some examples

Match these patterns to the word examples

```
>>> d
```

data

coding

```
>>> ^g
```

green

gpn

```
>>> ^.l+
```

```
>>> ^(b|a)+
```

algorithm

abbreviate

illogical

bash

Some examples

Match these patterns to the word examples

```
>>> d
```

data

coding

abbreviate

```
>>> ^g
```

green

gpn

```
>>> ^.l+
```

algorithm

illogical

```
>>> ^(b|a)+
```

bash

Some examples

Match these patterns to the word examples

```
>>> d
```

data	**coding**

```
>>> ^g
```

green	**gpn**

```
>>> ^.l+
```

algorithm	**illogical**

```
>>> ^(b|a)+
```

bash	**abbreviate**

How to actually use regex

There are two main functions we will be using.

1. `re.search(r"pattern", string_to_search)`

How to actually use regex

There are two main functions we will be using.

1. `re.search(r"pattern", string_to_search)`

This will return none if the pattern is not found and a class if it is.

How to actually use regex

There are two main functions we will be using.

there must be an “r” before the pattern string to ensure python registers the regex pattern correctly

```
1.  re.search(r"pattern", string_to_search)
```

This will return none if the pattern is not found and a class if it is.

How to actually use regex

There are two main functions we will be using.

1. `re.search(r"pattern", string_to_search)`

This will return none if the pattern is not found and a class if it is.

2. `re.findall(r"pattern", string_to_search)`

How to actually use regex

There are two main functions we will be using.

1. `re.search(r"pattern", string_to_search)`

This will return none if the pattern is not found and a class if it is.

2. `re.findall(r"pattern", string_to_search)`

This will return a list of all the matches to the pattern

A useful website

If you want to learn more regex building blocks or test your regex expressions in a way that will explain the outcome... We suggest looking at

https://regex101.com/

Project time!

You now know all about Regex!

Let's put what we learned into our project

Try to do the Extensions!

The tutors will be around to help!