LEARN TO CODE

CoderDojo

LISTS

Card 1 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

1 What if you wanted to store a bunch of information? Say, for example, the names of everyone at your Dojo? For this, you use a **list**:

```
list_of_names = ["Alice","Bob","Chris"]
list_of_numbers = [1,2,3,4,5]
```

2 **Lists** are really useful. To start with, it's easy to print out the whole **list**.

```
names = ["Chis","Alice","Bob","Emma","Danny"]
print(names)
```

3 How about counting how many items are in the **list**? That's easy! Use the **len( )** function. It's short for length. Update your code and try it:

```
names = ["Chis","Alice","Bob","Emma","Danny"]
names_count = len(names)
print("There are "+str(names_count)+" names.")
```

4 Nice! Now, what if you want to get the first name on the **list**? You use square brackets after the variable name, with the position of the item you want into the brackets. **Notice that computers count from 0!**

```
print("The first name is "+ names[0]+".")
```

CoderDojo

LISTS

Card 1 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

5 You can create a copy of a **list** (so, for example, you can change one without changing the other), like this:

```
copy_of_names = list(names)
```

6 What if someone new needs to be added? You can do that with the **append** method. Append basically means "stick on the end". It works like this:

```
names.append("George")
```

Try adding a few names to your **list** now. Use the names of some of the people around you.

7 Ok, now how do you take a name off the **list**? Again, Python has made this pretty easy for you. You just use the **remove** method, like this:

```
names.remove("Bob")
```

8 Finally, it's possible to totally reverse the order of a list by using the **reverse** method, like this:

```
names.reverse()
```

LEARN TO CODE

CoderDojo

FOR LOOPS

Card 2 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

1 This card will show you how to make **for** loops. These are important if you want to do something a certain number of times, or to everything on a list. You could use this to **print** out a countdown, or display all the messages in a chat conversation. This card has a few examples you can create as programs to try it for yourself.

2 This one will **print** out the numbers from 1 to 10. We do this using the **range** function, which gives us all the **integers** between the two we put into it. The first number is included, but it stops before reaching the last one. The name you use for the "current value" **variable** in a **for** loop doesn't strictly matter. When we're just counting like this, programmers often use the letter **i**, which is short for **integer**: the kind of number you're counting. Here, we've done something a little more fun.

```
for hamster in range(1, 11)
    print(str(hamster))
```

The way this loop works is:

- At the start of the loop, take the first number that hasn't been used by the loop yet (on the first run through it'll be 1, then 2)
- Store that number in a variable called **hamster**
- For the rest of this run through the loop, **hamster** has that value
- Once that run is finished, start the loop again, unless there are no more numbers

CoderDojo

FOR LOOPS

Card 2 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

3

This one says hello to everyone named on a **list** of names. With the right code, you could use this to message everyone you know!

```
for name in names
    print("Hi "+name+"!")
```

The way this loop works is:

- At the start of the loop, take the first name that hasn't been used by the loop yet
- Store that name in a variable called **name**
- For the rest of this run through the loop, **name** has that value
- Once that run is finished, start the loop again, unless there are no more names

CoderDojo

DICTIONARIES

Card 3 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

1

Lists are very useful, but there are some things that it's very hard to do with them. Let's try to keep some info pets. You could do this:

```
pet_names = ["Fluffy", "Spot", "Felix"]
pet_type = ["Rabbit", "Dog", "Cat"]
pet_size = ["Small", "Medium", "Large"]
pet_eats = ["Carrots", "Bones", "Fish"]
```

But, this makes it hard to add things and means you can't **sort** them, since they're not connected.

2

We can solve this problem by using another new type of **variable**, closely related to the **list**: the **dictionary**. The you can look up the **values** in the dictionary with the **key**. Try it in your code!

```
 fluffy = {
     "name" : "Fluffy",
     "type" : "rabbit",
     "size" : "small",
     "eats" : "carrots"
 }
 # And so on for Spot and Felix, but I have to fit
this on a card!
 # We can get information out by passing the right
keys.
 print(fluffy["name"]+" is a "+fluffy["size"]+"
"+fluffy["type"]+" who likes to eat
"+fluffy["eats"]+".")
```

CoderDojo

DICTIONARIES

Card 3 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

3

You may be thinking "If I had to print out all the pets, I'd be doing the same thing over and over agin. I should be able to use a **for** loop!" You can, but you need to put the **dictionary** in a **dictionary** (or a **list**):

```
# Make an empty dictionary
pets = { }
# Adding a pet to the dictionary is simple
pets["fluffy"] = {
  "name" : "Fluffy",
  "type" : "rabbit",
  "size" : "small",
  "eats" : "carrots"
}
pets["spot"] = {
  "name" : "Spot",
  "type" : "dog",
  "size" : "large",
  "eats" : "bones"
}
pets["felix"] = {
  "name" : "Felix",
  "type" : "cat",
  "size" : "medium",
  "eats" : "fish"
}
# Now you can go through these pets with a loop.
# Note: you use two sets of square brackets.
for pet in pets:
 print(pets[pet]["name"]+" is a "+pets[pet]
["size"]+" "
 +pets[pet]["type"]+" who likes to eat "+pets[pet]
["eats"]".")
```

CoderDojo

FUNCTIONS

Card 4 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

1 You've seen in the last few cards how programmers use loops to save themselves from having to write code to do the same thing over and over. There's another way you can re-use the same code: You can use **functions**.

Functions let you give a label to a piece of code and "call" it using that label from other places in your program. **Print** is a **function** and so are the **list functions** for sorting, appending and removing items.

2 You create a function by using the word **def**. If you want to make a function that says hello to people, you can do it like this:

```
def greet(name):
    print("Hello "+name+"!")
```

This **function** is called "greet" and you must pass it a **variable** called **name** when you call it. Here's an example of how you'd use it:

```
def greet(name):
    print("Hello "+name+"!")
# You can have any amount of code between them, but
the function has to be above the place where you
call it.
greet("Bob")
```

Functions are particularly useful where you need to do the same thing in several different places in your code.

CoderDojo

FUNCTIONS

Card 4 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

3

You can do a lot more in a **function** than just speed up a little bit of code. Here's a more useful example, where we use an "add_person" **function**. Try running this code and using it.

```
people = []
# Create a function to put people on the list
def get_person():
  # Collect information about the person
  person_name = input("What is their name?")
  person_age = int(input("How old are they?"))
  # Turn that information into a dictionary
  person = {
    "name" : person_name,
    "age" : person_age
  }
  # Give back the person
  return person

# Here's a short program using it
person_count = int(input("Add how many people?"))
for count in range(person_count):
  people.append(get_person())
print("You've added "+str(person_count)+" people!")
print("Here they are: "+str(people))
```

This **function** doesn't take any inputs, asks a few questions and then **returns** the result as a "person" **dictionary** which is **appended** to a **list** of "people".

LEARN TO CODE

CoderDojo

FILES

Card 5 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

1 So far, your programs have only taken input from users by asking them questions directly. You also haven't been able to save anything form one running of your program to the next. With this card, you can fix that! You can use files to **read** and **write** data into and out of your program, keeping things like those lists of favourite pets between uses of the program.

2 First, you're going to need a file to work with. Go to dojo.soy/py-file and save the file there (or its contents) to the same location as your Python program.

3 Once the file is in place, you can read it into your program. Try this to read the contents of the file and print them out without changing them:

```
# Create a variable that holds the file.
# The "r" means you will read from the file
# You would use a "w" to write to it
file = open("python-example.txt","r")
file_text = file.read()
print(file_text)
```

CoderDojo

FILES

Card 5 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

4 When writing out to a file you have two choices for how to do so:

- Start a new file with that name (and delete any old ones) then write to it. This is standard **writing** and we use a **w** in the **open function**. You would do this if you wanted to either edit the existing contents of a file, or make a whole new one.
- Find an existing file with that name (or create a new one if one doesn't exist) and write onto the end of it. This is called **appending** and we use an **a** in the **open function** to indicate it. This can be useful when you want to add to an existing record. For example, you could use it to keep a record of a chat conversation. To get an idea of how this works, try both of them and open the **output-example.txt** file after each.

Write

```
file = open("output-example.txt","w")
# You can, of course, use a variable for the
file name, or the output text
file.write("Hello everyone. Now I can write to
files!")
```

Append

```
file = open("output-example.txt","a")
# Note that the text starts with "\n".
# This makes sure that what you're appending
starts on a new line.
# It's as though you hit the enter key wherever
that appears in the text.
file.write("\nHello everyone. Now I can write to
files!")
```

LEARN TO CODE

CoderDojo

CHALLENGE!

Card 6 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

1 So now you've learned about:

- **lists**: Lists of variables that are easier to sort, order and loop over
- **dictionaries**: A collection of variables with easy to access labels
- **for** loops: Repeat some code a number of times, or on a list
- **Functions**: Re-use your code, or someone else's
- Files: Read from, write and append to a text file

2 Go to dojo.soy/py-pal and save the file there to your computer. You're going to write a program that:

- **Reads** in the contents of the file you downloaded
- Looks at whats on each line and checks **if** the line is what's called a **palindrome** (spelt the same backwards and forwards, ignoring punctuation like commas, periods, etc.) For example: "Taco Cat"
- **Writes** the lines into two files: the lines that are palindromes and the lines that aren't

Here are a few things you'll need to know:

3 Split a **file** into lines and, in this case, **print** them. The **rstrip("\r\n")** function removes the new line instructions at the end of each line.

```
my_file = open('py-pal.txt','r')
file_lines = my_file.readlines()
for line in file_lines:
  print(line.rstrip("\r\n"))
```

CoderDojo

CHALLENGE!

Card 6 of 6

I'm Learning: Python

Produced by the CoderDojo Foundation // @CoderDojo

4 Make all the letters in a **string** the same **case** (lowercase or UPPERCASE), as Python does not see **a** and **A** as the same letter.

```
my_text = 'Hello EVERYone'
lower_my_text = my_text.lower()
upper_my_text = my_text.upper()
```

5 Check if a an item is, or is not, on a list

```
my_pets = ['cat','dog','rabbit']

if('rabbit' in my_pets):
   print("I have a bunny!")

if('snake' not in my_pets):
   print("I don't have a snake.")
```

6 Turn a **string** into a **list** of the **characters** that make it up

```
my_string = "Hello world!"
my_list = list(my_string)
```

7 Now, you have everything needed to solve this problem. It's a tough one though, so you can check with my solution at dojo.soy/py2ans if you get stuck. Once you're done, let us know at dojo.soy/py-intermediate what you thought!