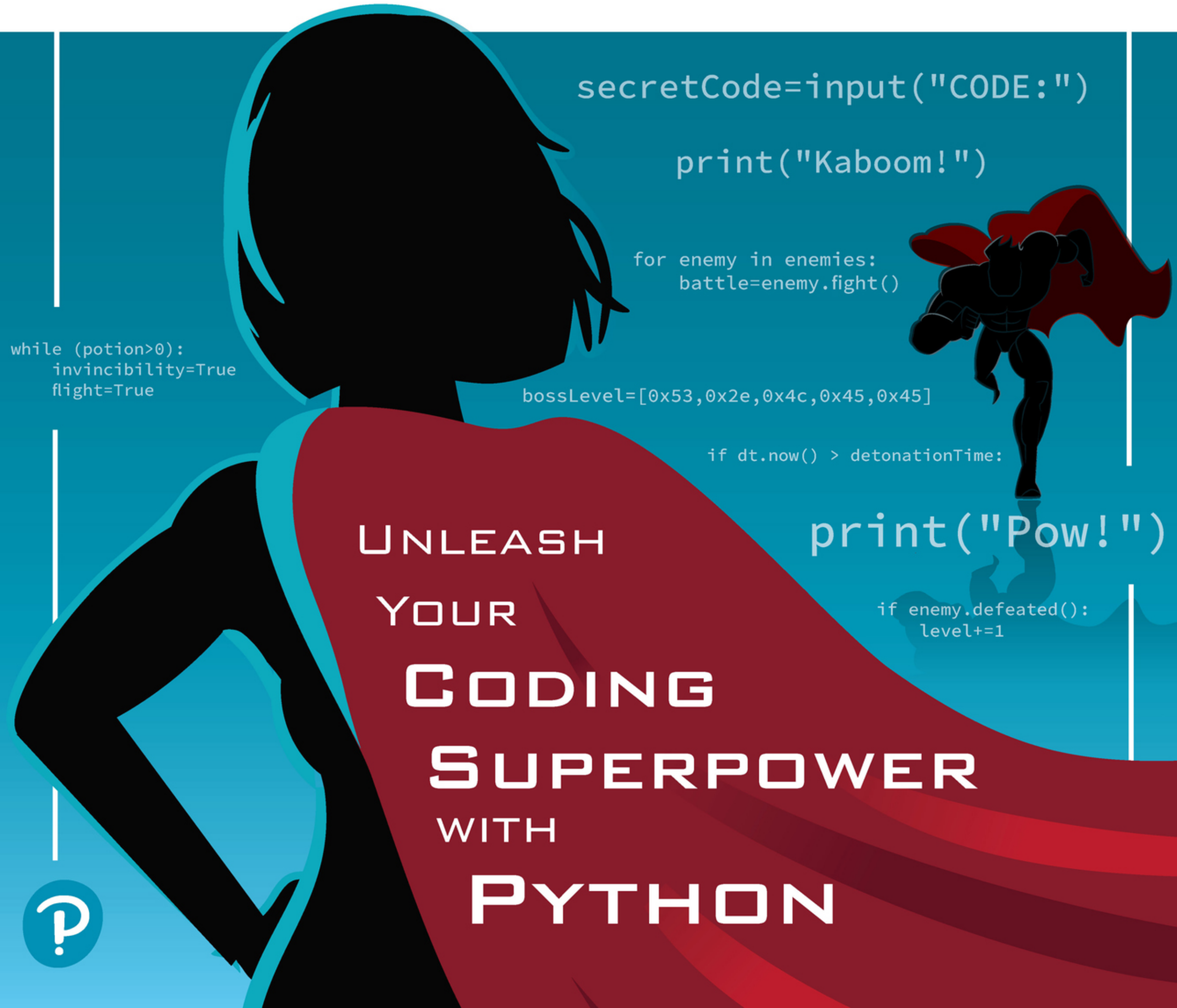
BEN FORTA & SHMUEL FORTA

# CAPTAIN CODE

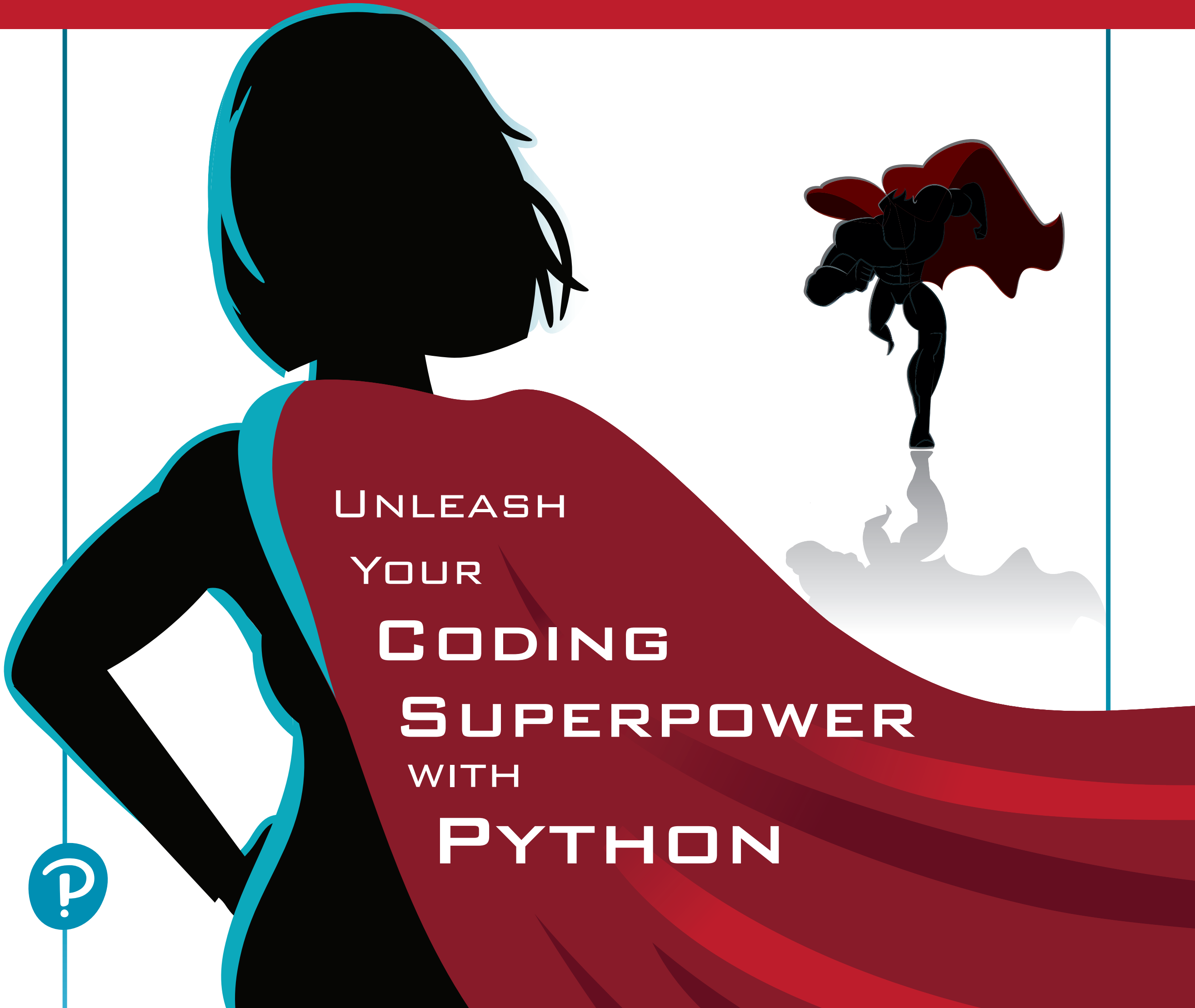


Unleash Your Coding Superpower with Python

BEN **FORTA** & SHMUEL **FORTA**

# CAPTAIN CODE

Unleash Your Coding Superpower with Python



**Longer Keys Are Better**

This type of encryption can be broken by looking for patterns and repeats. A short key will result in lots of repeats, and a longer key in fewer ones, so the longer your key, the harder it will be for your nemesis to decrypt your secret plans. You have been warned!

But what if your text is longer than the key? Say you were encrypting the text `Hello World`. The key has `6` digits in it, we need `11` digits (because the space value has an ASCII value, too). What to do?

The answer is reuse the key. If the key is 6 digits long, we use these 6 digits for the first 6 letters to be encrypted. And then we start over: For letter 7, we use the first digit in the key, for letter 8 the second, and keep reusing the key over and over as needed.

How do we figure out which digit to use? We use division and look at the remainder. For the 8th character (we need the second number from the key), we just divide the character index (8, as this is the 8th character) by the key length (6), and the remainder is 2. In Chapter 3 we introduced the modulus operator (`%`), which finds a remainder. It is used like this:

```
print(8%6)
```

Here `8` is divided by `6`, and the remainder is `2`.

Using modulus, we can always divide the character position (`8` for the 8th character, `42` for the 42nd, and so on) by the key length, and the remainder will point to a valid key digit to use.

## Encryption Code

Ok, here's the code for file `Encrypt.py`:

```
# ASCII range of usable characters - anything out of this range
could throw errors
asciiMin = 32   # Represents the space character - " "
asciiMax = 126  # Represents the tilde character - "~"
```

```
# Secret key
key = 314159    # Top secret! This is the encryption key!
key = str(key)  # Convert to string so can access individual digits

# Get input message
message = input("Enter message to be encrypted: ")

# Initialize variable for encrypted message
messEncr = ""

# Loop through message
for index in range(0, len(message)):
    # Get the ASCII value for this character
    char = ord(message[index])
    # Is this character out of range?
    if char < asciiMin or char > asciiMax:
        # Yes, not safe to encrypt, leave as is
        messEncr += message[index]
    else:
        # Safe to encrypt this character
        # Encrypt and shift the value as per the key
        ascNum = ord(message[index]) + int(key[index % len(key)])
        # If shifted past range, cycle back to the beginning of the
range
        if ascNum > asciiMax:
            ascNum -= (asciiMax - asciiMin)
        # Convert to a character and add to output
        messEncr = messEncr + chr(ascNum)

# Display result
print("Encrypted message:", messEncr)
```

Save and run this code. It will ask you for a message to encrypt and will then display the encrypted message. It doesn't decrypt; we'll get to that next.

So how does this work?

Not all ASCII characters print well, so to be safe, you define the range of characters you want to use, like this:

```
asciiMin = 32   # Represents the space character - " "
asciiMax = 126  # Represents the tilde character - "~"
```

Next comes the key:

```
# Secret key
key = 314159    # Top secret! This is the encryption key!
key = str(key)  # Convert to string so can access individual digits
```

`key` is the numeric encryption key. This one has six digits, though yours can be longer or shorter. The key here (hee hee, bad pun, sorry) is that you must have the same key to encrypt and decrypt the text.

We need to use each digit in the key individually. Remember? That's how we can encrypt each character with a different key digit. To do this, we convert the key to a string, so `314159` becomes `"314159"` because getting characters from a string is super easy. It's much like getting them from a list. Remember those?

Next, the code asks for the text to be encrypted using an `input()` function. You're very familiar with this one.

Then this code is used:

```
messEncr = ""
```

This creates an empty string variable named `messEncr` (for *message encrypted*). The code is going to encrypt the text one character at a time, and as it does so, the encrypted character will be added to this variable.

**`str()` and `int()`**

`str()` is the opposite of the `int()` function you saw in Chapter 4. `int()` converts a string to a number, and `str()` converts numbers back to strings. We'll look at `int()` in more detail in Chapter 7.

Now we loop through message:

```
for index in range(0, len(message)):
```

Here we use a for loop that loops from `0` to the length of the text. How does the loop know how long the text is? Once again, we can use the `len()` function for that. If the text is 10 characters long, `len()` returns `10`, so the range for the loop will be `range(0, 10)`, meaning loop from 0 to 9, exactly what we need. Within each iteration, the variable `index` will contain the iteration number: `0` the first time, `1` the second, and so on.

The code indented under the `for` statement will be executed once per character to be encrypted. At the start of each loop, we need to get the ASCII value for the letter being processed, like this:

```
char = ord(message[index])
```

`message[index]` lets us access a single character. `index` is `0` on the first loop, so on the first iteration, `message[index]` will return the first character. On the next iteration, it will return the second. `ord()` gets the ASCII code for the number, and that code is saved to the variable `char`.

The next `if` statement checks to see if the ASCII code for this character is within the safe range. If not, we don't encode it. If yes, this code gets executed:

```
ascNum = char + int(key[index % len(key)])
```

This code does the actual encryption. `index` is the current character number (set by the `for` loop). `index % len(key)` divides the `index` by the length of the key, giving us a remainder, which is the key digit to use. That gets added to `char` (the current ASCII code), and the result is saved in `ascNum`. So if, for example, the loop is currently at index `9`, and the key has six digits, `index % len(key)` will become `9 % 6`, which returns `3` (the remainder), and index `3` of the key will be used.

The encoded character then gets appended to `messEncr`, like this:

```
messEncr = messEncr + chr(ascNum)
```

chr(ascNum) converts the newly calculated encoded character to a string, and that gets appended to messEncr. (Remember that adding strings concatenates them.)

As we mentioned previously, some ASCII characters don't print, and so we need to make sure to exclude these. This code checks to ensure that the encoded character is within the safe range, and if it isn't, it shifts to a safer value:

```
if ascNum > asciiMax:
    ascNum -= (asciiMax - asciiMin)
```

And finally, a print() is used to display the encrypted text.

That does it. Enter text, and the program will encrypt it using the digits in the secret key. If you send someone a message, they'd need the matching key to read it. You can use different keys for different people (that way they won't be able to read each other's messages).

## Decryption Code

Great. But how do you decrypt the encrypted messages? The process is actually exactly the same. It's so similar, in fact, that you can use the same Encrypt.py file and just make a few changes.

**TIP**

**Use Save As** If you use the VS Code Save As option (in the File menu) to save Encrypt.py as Decrypt.py, you have two files that are identical. Try it!

Click on Decrypt.py and we'll make a few changes. First, change the input() so the prompt is correct:

```
message = input("Enter message to be decrypted: ")
```

Next, find the line that does the actual encryption, which looks like this:

```
ascNum = char + int(key[index % len(key)])
```

Recall that when we encrypt, we add a key digit. To decrypt, all we need to do is subtract the same digit. So, change the code to this:

```
ascNum = char - int(key[index % len(key)])
```

The + is changed to a `-`.

Next, look at the `if` statement right below the line you just edited. It checks to see that we haven't gone above the allowed range and subtracts the change, if needed. We need to reverse that so it looks like this:

```
if ascNum < asciiMin:
    ascNum += (asciiMax - asciiMin)
```

In the `if` statement, the `>` gets changed to a `<`, and in the assignment, the `-=` becomes `+=`. Now if the decoding process creates a number below the range, we can fix that.

Some of the comments will be wrong, best to fix those, too.

That's it! Now you can encrypt and decrypt messages, so long as both parties have the same key. And all made possible by some simple `for` loops.

Oh, using the key `314159`, were you able to decrypt the encrypted text we showed you earlier in this chapter?

## CHALLENGE 6.2

As you have seen, `Encrypt.py` and `Decrypt.py` are almost identical. In truth, they should have been the same program. We just separated them to make the code a little simpler.

But, you can improve this. Create a program that both encrypts and decrypts. It will need to prompt the user for the action—something like this:

```
action = input("Encrypt or decrypt? Enter E or D: ")
```

Then, in your code, you can use `if` statements to select the encrypt or decrypt versions of the code, based on `action` being `E` or `D`.