Starting chapter 3

Strings

- Chapter 3's problem context is cryptography, but mostly it is about strings and related ideas
- Strings are basically sequences of characters
- A string literal is enclosed in quotes (`''` or `""` in Python):

  ```
  'hello' == "hello"  >>> True
  ```
- Actually objects of a Python class named `str`

  ```
      type('kitty')    >>> <class 'str'>
  ```
- Can assign names like any other type of object

  ```
      message = "Don't be late!"
      print(message)   >>> Don't be late!
  ```
- Lots of built-in functions work for string objects, and `class str` has useful operators and methods too

Functions `chr(n)` and `ord(c)`

- Characters are numbers in memory – e.g., ASCII codes
- For example, `'A'` has code `65` in ASCII
  - Use `ord` function to verify: `ord('A')  >>> 65`
  - Notice `'A'` is not same as `'a'`: `ord('a') >>> 97`
- Conversely, can find character associated with a particular code using `chr` function

```
chr(65)    >>> 'A'
```

- Can manipulate numbers to process characters

```
chr( ord('a') + 3)   >>>  'd'
```

- Notice digit characters have codes too – might surprise:

```
ord('6')   >>>  54
```

Try it!

A simple substitution cipher

- Just reverse order of characters in alphabet

```
def encrypt(message):
    result = ''    # start with empty result
    for c in message:
        nc = ord(c)  # get order; reverse on next line
        nr = ord('a') + ord('z') - nc
        result = result + chr(nr)    # accumulate
    return result

>>> encrypt("abcdefghijklmnopqrstuvwxyz")
'zyxwvutsrqponmlkjihgfedcba’
```

- Same function decrypts, by the way:

```
>>> encrypt('zyxwvutsrqponmlkjihgfedcba')
'abcdefghijklmnopqrstuvwxyz'
```

- What happens if `encrypt("CAT")`? How to fix?

String + and * operators

- + is the concatenation operator

```
“really ” + “cool” >>> 'really cool'
```

- Another accumulator pattern example:

```
cheer = “”    # start with “empty string”
for i in range(4):
    cheer = cheer + ‘go ’
print(cheer) >>> go go go go
```

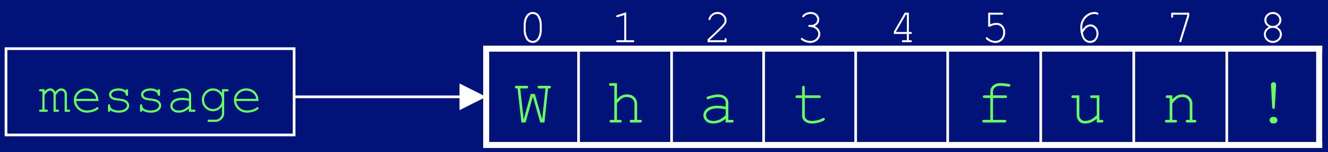
- Shortcut: * – the string repetition operator

```
'go ' * 4  >>> 'go go go go '
```

String indexing: `[]` operator

- Each character in a string has a position
  - First position is 0 – means 0 *offset* from the beginning

    `message = “What fun!”`



  - So `message[0]` is ‘`W`’, `message[1]` is ‘`h`’, …
- Python strings are immutable

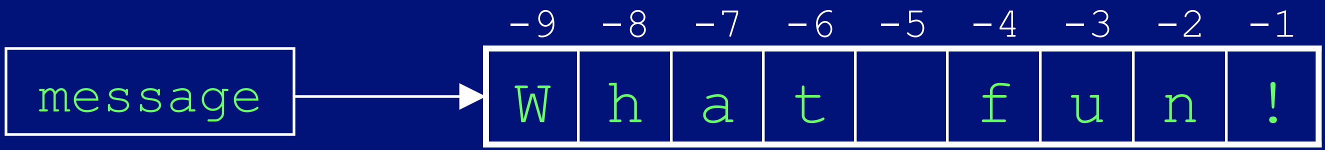
    `message[8] = “?”` # illegal operation
- But okay to reassign name to new string:

    `message = “What? Fun?!”`

More string indexing

- Can also index from end of string
  - Last position is -1 (Note: this is unique to Python.)

    `message = “What fun!”`



  - So `print(message[-3])` prints `u` in this case
- Use built-in `len` function to know length of string

  `len(message)` → `9` in this case
- So index range is 0 to length-1, or -1 to –length

  `message[0] == message[-len(message)]` → `True`

Index "slicing"

- A *range* of string indices – a.k.a. substring
  - Slice operator, `[:]` – [*first index* : *1 past last*]
- E.g., if `name = "Mike"`

  ```
  name[1:3] >>> 'ik'
  ```

  - Omit index after colon – means "to the end"

  ```
  name[2:]  >>> 'ke'
  ```

  - Omit index before colon – means "from beginning"

  ```
  name[:2]  >>> 'Mi'
  ```

String methods

Try these things to learn.
P.S. Try `in` keyword too.

- Actually defined in `class str`
  - Are many – see Table 3.2 and try `help(str)`
- Definitely worth playing with

```
s = "Row, row, row your boat"
s = s + " gently down the stream."
s.count("ow") >>> 4
s.find("row") >>> 5    # first index only
s.find(“banana") >>> -1    # means not found
s[:13].upper() >>> 'ROW, ROW, ROW'
s[:13].upper().replace('R','GR')
  >>> 'GROW, GROW, GROW'
```

Writing string functions

- Can't actually change a string – so usually create a new one to return
  - Often means using accumulator pattern for strings:

    `result = ""   #` initial value is empty string

    - Then inside loop: `result = result + ...`
- Sometimes can find a quicker way by slicing – but *watch for bugs like in listing 3.6* (p. 104):

```
def removeChar(string, idx):
    return string[:idx] + string[idx+1:]
```

  - Okay except when `idx = -1` (How to fix?)
- See/try other examples (and without bugs):
  - Listings 3.8 (removeDupes) and 3.9 (removeMatches)

Getting text from the user

- Page 99 of the text has a "by the way" section about getting input text from a user!
- Simplest way is with built-in `input` function:

```
answer = input("Enter some text: ")
```

  - Parameter is the "prompt" – tells user to enter text
  - User's text is returned as a string

Try it!

- Q. What if you want to get a number?
- A. Create `int` or `float` object from string

```
number = float(answer)     # might fail though
```

Better encryption techniques

- Transposition (a.k.a. rail fence) cipher
  - Like most encryption techniques, it has a related decrypt function: Listings 3.2 and 3.3
- Substitution cipher
  - Substitutes letters based on rearranging alphabet according to a key (like a password)
  - Note: subject of cryptanalysis in Chapter 8
- Vignère cipher
  - Based on Vignère square – basically, substitution that varies letter by letter: Listings 3.11 and 3.12 (with helper functions from Listing 3.1)

Next

Lists and other Python collection types