Strings

James Gain
Department of Computer Science
University of Cape Town
jgain@cs.uct.ac.za

Problem 1 Introduction

- Write a program to print out the reverse of a sentence.
- For example:
  - Computer becomes retupmoC
- Use first principles - i.e., process the string character-by-character.

Strings

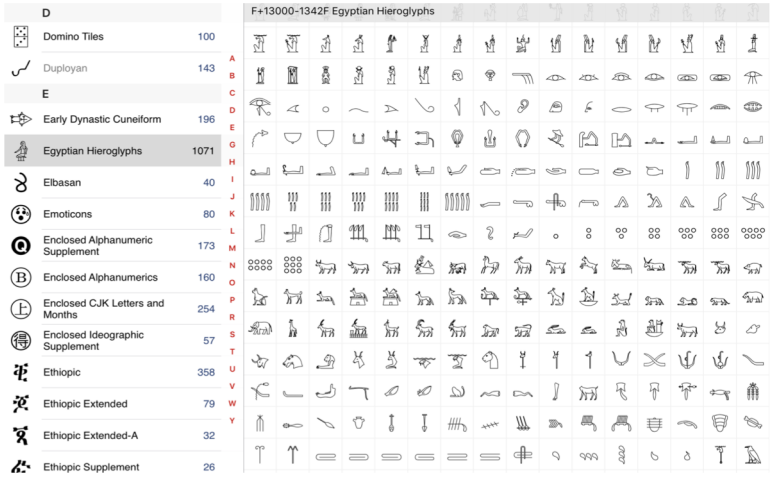
- Basically a sequence of characters (letters, digits, symbols).
  - e.g., `"howzit %*&_^96"`
- String literals are enclosed in single ' or double " quotes.
- Use escape characters within strings if necessary.
- The data type of strings is **string**.
- All strings are objects so have methods for various useful functions.

How Python stores strings

- Strings are sequences of characters.
- A character is internally a single number representing some symbol.
- The mapping from numbers to symbols is called Unicode - it is a standard for electronic data.
- Unicode has thousands of symbols defined to cater for all living languages!

Symbol	...	A	B	C	...	a	b	c	...
Unicode number	...	65	66	67	...	97	98	99	...

And some non-living languages



Internal data vs Input/Output

- What the computer stores:

72	101	108	108	111	32	87	111	114	108	100

- What the user sees on the screen:

H	e	l	l	o		W	o	r	l	d

Processing strings

- + to join strings together
  - Example: `"hello" + " " + "world"`
  - Produces: `"hello world"`
- * to multiply a string
  - Example: `"hello"*2`
  - Produces: `"hellohello"`
- len () for length of a string
  - len(s)
  - Example: `len("hello")`
  - Produces: `5`

Indexing a string

- An index is used to read a single character from a string.
- We can read only - we cannot change characters.
- Syntax:
  - `a_string[i]`
  - returns the single character in position i
- Example
  - `"Hello World"[4]`
  - Produces: `"o"`

string	H	e	l	l	o		W	o	r	l	d
positions	0	1	2	3	4	5	6	7	8	9	10

Iterating over string characters

- We can iterate over the characters of a string using **for**.
  - Example:

```
word = "Hello"
for a_char in word:
    print (a_char*2,end="")
```

  - Output:

```
HHeellllloo
```

- Alternatively, we can iterate over the position numbers:

```
for index in range (len(word)):
    print (word[index]*2,end="")
```

Converting to/from numbers

- `ord("a")`
  - return the Unicode number for 1-character string "a"
- `chr(97)`
  - returns the 1-character string with Unicode symbol 97
- `int("1234")`
  - returns the integer value 1234
  - note: we can also use eval() and float()
- `str(1234)`
  - returns the string value "1234"

Problem 1

- Write a program to print out the reverse of a sentence.
- For example:
  - Computer becomes retupmoC
- Use first principles - i.e., process the string character-by-character.

Problem 2

- Print out a table of Unicode numbers and corresponding symbols.
- Try the first 1000 or some user-selectable range.

Problem 3

- Stylistically, in electronic communication, CAPITAL letters are considered to be the equivalent of shouting. However, many people consider electronic shouting to be rude.
- Write a program to convert a sentence into all lowercase.
- Use first principles - i.e., process the string character-by-character.

Problem 4

- Write a program that encrypts and decrypts strings made up of alphabetic characters by converting them to and from Greek characters
  - Ignore everything that is not a letter
  - Use the underlying Unicode numbers

Problem 5 Introduction

- Suppose we have a variable containing:
  - `"the quick brown fox jumps over the lazy dog".`
- Write a program to extract the colour of the quick fox from the sentence using only string manipulations. Make sure your program will work even if the string is different, as long as there is a quick something fox in it!
- Can you modify the program to change the colour to something else?

String Slicing

- Slicing returns a sequence of characters from a string.
- Syntax:
  - `a_string[start:stop:step]`
  - returns characters from start, ending before stop, step characters apart.
- Notes:
  - if step is negative, string is processed from back to front
    - last character is position -1, then -2, etc.
  - step is optional (and you can omit the :) - assumed to be 1
  - start is optional - assumed to be 0 (or -1 for negative step)
  - stop is optional - assumed to be len (or -len-1 for negative step)

string		H	e	l	l	o		W	o	r	l	d	
positions		0	1	2	3	4	5	6	7	8	9	10	11
neg. positions	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

String Slicing Quiz

- What string is generated from the following expressions?
  - Assume greet =“Hello Bob”

```
greet[0:3]
greet[5:9]
greet[:5]
greet[5:]
greet[:]
```

More String Slicing Examples

```
a = "Jabberwocky”
b = a[::2] # b = 'Jbewcy'
c = a[::-2] # c = 'ycwebJ'
d = a[0:5:2] # d = 'Jbe'
e = a[5:0:-2] # e = 'rba'
f = a[:5:1] # f = 'Jabbe'
g = a[:5:-1] # g = 'ykcow'
h = a[5::1] # h = 'rwocky'
i = a[5::-1] # i = 'rebbaJ'
j = a[5:0:-1] # 'rebba'
```

String Slicing

- Slicing returns a sequence of characters from a string.
- Syntax:
  - `a_string[start:stop:step]`
  - returns characters from start, ending before stop, step characters apart.
- Notes:
  - if step is negative, string is processed from back to front
    - last character is position -1, then -2, etc.
  - step is optional (and you can omit the :) - assumed to be 1
  - start is optional - assumed to be 0 (or -1 for negative step)
  - stop is optional - assumed to be len (or -len-1 for negative step)

string		H	e	l	l	o		W	o	r	l	d	
positions		0	1	2	3	4	5	6	7	8	9	10	11
neg. positions	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

String Slicing Quiz 2

- Given:
  - `a = "impossible"`
- Find two ways to extract each of the following substrings:
  - `b = "possible"`
  - `c = "ioie"`

Objects

- Objects are a special data type, including both the actual data and functions (called methods) that operate on the data. All Python strings are objects.
- Examples of string functions/methods include:
  - count, find, join, lower, ...
- For non-object functions, we specify the string as a parameter:
  - `len(s)`
- For object functions, we call the function on the string using a dot and then the function name (aka dot-notation):
  - `s.lower()`

String methods

- `s.count(search)`
  - returns integer number of *search* strings found within s
- `s.find(search)`
  - returns position of first *search* string found within s
- `s.lower()`
  - returns lowercase version of s
- `s.replace(old, new)`
  - returns version of s with every occurrence of old replaced with new
- `s.upper()`
  - return uppercase version of s

Poll: String Methods

- What is the output of the following Python code:

```
magic = "Abracadabra"
newmagic = magic.replace("abra", "cobra")
print(newmagic.upper())
```

- COBRACADCOBRA
- Abracadcobra
- cobracadabra
- ABRACADCOBRA

Problem 5

- Suppose we have a variable containing:
  - `"the quick brown fox jumps over the lazy dog".`
- Write a program to extract the colour of the quick fox from the sentence using only string manipulations. Make sure your program will work even if the string is different, as long as there is a quick something fox in it!
- Can you modify the program to change the colour to something else?

String Formatting

- Use a format language to specify a template and expressions to fit into the template.
- General syntax:
  - <template string>.format (<var1>, <var2>, …)
- Python has multiple formatting approaches – this is one!

String Formatting Examples 1

- Print 2 variables in order
  - `"{0} {1}".format ("one", "two")`
  - `'one two'`
- Reverse order
  - `"{1} {0}".format ("one", "two")`
  - `'two one'`
- Left-aligned in fixed width
  - `"{0:<20}".format ("one")`
  - `'one                 '`

String Formatting Examples 2

- Right-aligned in fixed width
  - `"{0:>20}".format ("one")`
  - `'                 one'`
- Centred in fixed width
  - `"{0:^20}".format ("one")`
  - `'        one         '`
- Floating point number rounding
  - `"{0:5.3f}".format (1.23456789)`
  - `'1.235'`

Summary - Basics

- Working with Python files and an IDE
- Structure of Python programs
  - Comments
  - Indentation
  - Variables and Data Types
  - Expressions
- I/O
  - `print()` and Escape sequences
  - `input()` and `eval()`

Summary - Selection

```
if boolean_expression:
   statement1
   statement2
else:
   statementa
   statement
```

- "if" ladder variant, with `elsif`
- Boolean expressions
  - Including `and, or, not`

Summary - Iteration

- Counter-controlled loops

```
for var in range(start, stop, step):
   statement1
   statement2
```

- Condition-controlled loops

```
while condition:
   statement1
   statement2
```

- Other aspects: inifinite loops, `break`, `continue`, `pass`, `for:-else`:

Summary - Strings

- Python is rightly famous for offering lots of string manipulation functionality
  - An underlying Unicode representation
  - Indexing and iteration
  - Slicing
  - String methods (find, replace, lower, upper, etc.)
  - Formatting