

Lecture 8 Strings

Objectives

- To understand the string data type and how strings are represented in a computer.
- To understand the basic idea of sequences and indexing as they apply to Python strings and lists.
- To get familiar with various operations that can be performed on strings through built-in functions and the string library.

- The most common use of personal computers is word processing.
- Text is represented in programs by the string data type.
- A string is a sequence of characters enclosed within quotation marks (") or apostrophes (').

```
>>> str1="Hello"
>>> str2='spam'
>>> print(str1, str2)
Hello spam
>>> type(str1)
<class 'str'>
>>> type(str2)
<class 'str'>
```

• Getting a string as input

```
>>> firstName = input("Please enter your name: ")
Please enter your name: John
>>> print("Hello", firstName)
Hello John
```

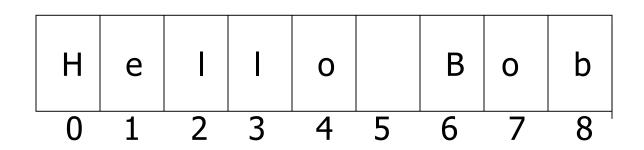
• Notice that the input is not evaluated. We want to store the typed characters, not to evaluate them as a Python expression, e.g. convert to int.

- We can access the individual characters in a string through indexing.
- The positions in a string are numbered from the left, starting with 0.
- The general form is <string>[<expr>] where the value of expr (i.e. an integer) determines which character is selected from the string.

```
    H
    e
    I
    I
    o
    B
    o
    b

    0
    1
    2
    3
    4
    5
    6
    7
    8
```

```
>>> greet = "Hello Bob"
>>> greet[0]
'H'
>>> print(greet[0], greet[2], greet[4])
H l o
>>> x = 8
>>> print(greet[x - 2])
B
```



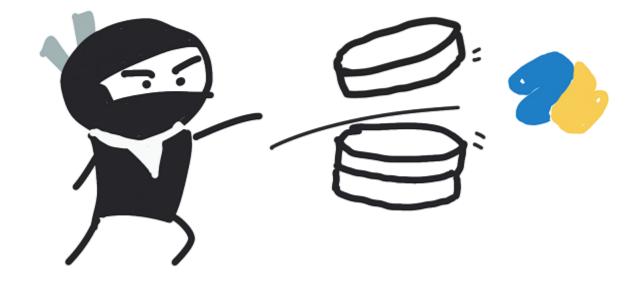
- In a string of *n* characters, the last character is at position *n-1* since we start counting with 0.
- We can index from the right side using negative indexes.

```
>>> greet[-1]
'b'
>>> greet[-3]
'B'
```

• Indexing returns a string containing a single character from a larger string.

• We can also access a contiguous sequence of characters, called a substring, through a process called slicing.

Slicing Strings



changhsinlee.com

- Slicing:<string>[<start>:<end>]
- start and end should both be ints
- The slice contains the substring beginning at position start and runs up to but does NOT include the position end.

 H
 e
 I
 I
 o
 B
 o
 b

 0
 1
 2
 3
 4
 5
 6
 7
 8

```
>>> greet[0:3]
'Hel'
>>> greet[5:9]
' Bob'
>>> greet[:5]
'Hello'
>>> greet[5:]
' Bob'
>>> greet[:]
                 This is same as greet
'Hello Bob'
```

- If either start or end expression is missing, then the start or the end of the string are used.
- Can we put two strings together into a longer string?
- Concatenation "glues" two strings together (+)
- Repetition builds up a string by multiple concatenations of a string with itself (*)

Operator	Meaning
+	Concatenation
*	Repetition
<string>[]</string>	Indexing
<string>[:]</string>	Slicing
len(<string>)</string>	Length
for <var> in <string>:</string></var>	Iteration through characters

L8 Strings - 14

```
>>> "spam" + "eggs"
'spameggs'
>>> "Spam" + "And" + "Eggs"
'SpamAndEggs'
>>> 3 * "spam"
'spamspamspam'
>>> "spam" * 5
'spamspamspamspam'
>>> (3 * "spam") + ("eggs" * 5)
'spamspamspameggseggseggseggs'
```

The function len() is used to return the length of string.

Simple String Processing

- Abbreviations of species names.
 - In a particular bioinformatics database, names of species are abbreviated to the first 3 letters of the first part (genus) and first 2 letters of the second part (species)
 - For example
 - Canis familiaris (dog) ⇒ CANFA
 - Pan troglodytes (chimp) ⇒ PANTR

Simple String Processing

```
# get genus and species names
genus = input("Please enter the genus: ")
species = input("Please enter the species: ")

SPECIES_CODE = genus[:3] + species[:2]
# Convert to upper case
SPECIES_CODE = SPECIES_CODE.upper()
```

- We'll be looking at string functions, including upper(), later
- This form of functions are called methods. Notice variable name followed by dot followed by function call.

String Representation

- Inside the computer, strings are represented as sequences of 1's and 0's, just like numbers.
- A string is stored as a sequence of binary numbers, one number per character.
- The mapping of characters to binary codes is arbitrary
 - as long as everyone uses the **same** mapping.

String Representation

- In the early days of computers, each manufacturer used their own encoding of numbers for characters.
- ASCII system (American Standard Code for Information Interchange) uses 127 characters using 8-bit (1 byte) codes
- Python also supports Unicode which maps 100,000+ characters using variable number of bytes

String Representation

- The ord function returns the numeric (ordinal) code of a single character.
- The chr function converts a numeric code to the corresponding character.

```
>>> ord("A")
65
>>> ord("a")
97
>>> chr(97)
'a'
>>> chr(65)
'A'
```

- Using ord and chr we can convert a string into and out of numeric form.
- The encoding algorithm is simple:
 get the message to encode
 for each character in the message:
 print the letter number of the character
- A for loop iterates over a sequence of objects, so the for loop over a string looks like:
 for <variable> in <string>

```
# text2numbers.py
#
      A program to convert a textual message into a sequence of
#
          numbers, utlilizing the underlying Unicode encoding.
def main():
    print("This program converts a textual message into a sequence")
    print ("of numbers representing the Unicode encoding of the
  message. \n")
    # Get the message to encode
    message = input ("Please enter the message to encode: ")
    print("\nHere are the Unicode codes:")
    # Loop through the message and print out the Unicode values
    for ch in message:
                                               Replace new line by comma
        print (ord (ch), end=",")
    print() # Go to new line at the end of code sequence
```

Shell >>> %Run encoder.py This program converts a textual message into a sequence of numbers representing the Unicode encoding of the message. Please enter the message to encode: fred Here are the Unicode codes: 102,114,101,100, >>>>

```
# A program to convert a textual message into a sequence of
# numbers, utilizing the underlying Unicode encoding.
Improved
def main():
    print("This program converts a textual message into a sequence")
    print ("of numbers representing the Unicode encoding of the
message. \n")
    message = input("Please enter the message to encode: ")
    print("\nHere are the Unicode codes:")
    for ch in message[:-1]:
        print(ord(ch), end=",")
    print (ord (message [-1]))
```

- We now have a program to convert messages into a type of "code", but it would be nice to have a program that could decode the message!
- The outline for a decoder:

```
get the sequence of numbers to decode
message = ""
for each number in the input:
    convert the number to the appropriate character
    add the character to the end of the message
print the message
```

- The variable message is an accumulator variable, initially set to the **empty string**, the string with no characters ("" or '').
- Each time through the loop, a number from the input is converted to the appropriate character and appended to the end of the accumulator.
- How do we get the sequence of numbers to decode?
- Read the input as a single string, then split it apart into substrings, each of which represents one number.

• The new algorithm:

```
get the sequence of numbers as a string, inString
split inString into a sequence of small strings (of digits)
message = ""  # Empty string
for each of the smaller strings:
    change the digits into the number they represent
    append the ASCII character for that number to message
print message
```

 $string \rightarrow list\ of\ digit\ strings \rightarrow list\ of\ numbers \rightarrow string$

```
"102,114" \rightarrow ["102","114"] \rightarrow [102,114] \rightarrow "fr"
```

- Strings have useful methods associated with them
- One of these methods is split. This will split a string into substrings based on a separator, e.g space.

```
>>> "Hello string methods!".split() This is a list.

['Hello', 'string', 'methods!'] More about them in the
```

next lecture

• This is the same as:

```
>>> a = "Hello string methods!"
>>> a.split()
```

• Split can use other separator characters other than space, by supplying the character as a parameter.

```
>>> "32,24,25,57".split(",")
['32', '24', '25', '57']
```

```
# numbers2text.pv
      A program to convert a sequence of Unicode numbers into
#
          a string of text.
def main():
    print ("This program converts a sequence of Unicode numbers into")
    print ("the string of text that it represents.\n")
    # Get the message to encode
    inString = input("Please enter the Unicode-encoded message: ")
    # Loop through each substring and build Unicode message
    message = ""
    for numStr in inString.split(','):
        codeNum = int(numStr) # convert the (sub)string to a number
        # append character to message
        message = message + chr(codeNum)
    print("\nThe decoded message is:", message)
```

- The split function produces a sequence of strings.
- Each time through the loop, the next substring is:
 - Assigned to the variable numStr
 - Converted to the appropriate Unicode character
 - Appended to the end of message.

Shell >>> %Run encoder.py This program converts a textual message into a sequence of numbers representing the Unicode encoding of the message. Please enter the message to encode: fred Here are the Unicode codes: 102,114,101,100, >>> %Run decoder.py This program converts a sequence of Unicode numbers into the string of text that it represents. Please enter the Unicode-encoded message: 102,114,101,100 The decoded message is: fred >>>

From Encoding to Encryption

- The process of encoding information for the purpose of keeping it secret or transmitting it privately is called *encryption*.
- *Cryptography* is the study of encryption methods.
- Encryption is used when transmitting credit card and other personal information through an insecure medium e.g. Internet.
- The Unicode mapping between character and number is an industry standard, so it's not "secret".

Encryption: Substitution Cipher

- In a simplistic way, if we replace the Unicode with some other code (that is only known to the sender and receiver) we will achieve encryption.
- This is called *substitution cipher*, where each character of the original message, known as the *plaintext*, is replaced by a corresponding symbol in the *cipher alphabet*.
- The resulting code is known as the *ciphertext*.
- This type of code is relatively easy to break.
- Each letter is always encoded with the same symbol, so using statistical analysis on the frequency of the letters and trial and error, the original message can be determined.

- There are a number of other string methods. Try them all!
 - str() Return a string representation
 - s.capitalize() Copy of s with only the first character capitalized
 - s.title() Copy of s; first character of each word capitalized
 - s.center(width) Center s in a field of given width

- s.count (sub) Count the number of occurrences of sub in s
- s. find (sub) Find the first position where sub occurs in s
- s.join(list) Concatenate list of strings into one large string using s as separator.
- s.ljust(width) Like center, $but \ s \ is \ left$ -justified
- s.rjust (width) Like ljust, but s is right-justified

- s.lower() Copy of s in all lowercase letters
- s.lstrip() Copy of s with leading whitespace removed
- -s.replace (oldsub, newsub) Replace occurrences of oldsub in s with newsub
- s.rfind(sub) Like find, but returns the rightmost position

- s.rstrip() Copy of s with trailing whitespace removed
- s.split() Splits into a list of substrings
- s.upper() Copy of s; all characters converted to uppercase

Summary

- We learned how strings are represented in a computer.
- We learned how various operations can be performed on strings.