THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 11
String processing and functions

Objectives

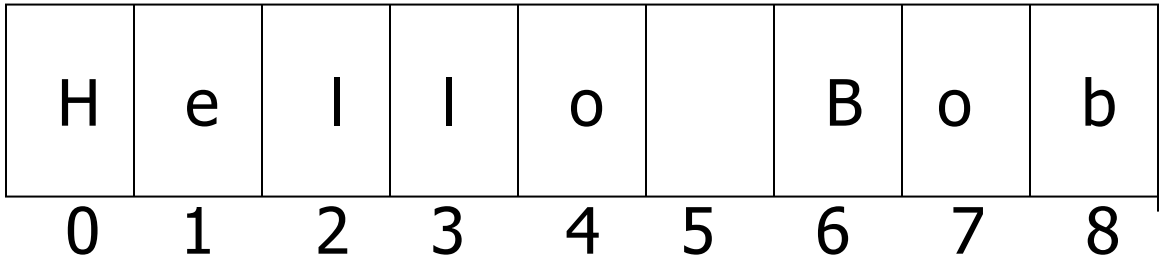
- Revision of string data type
- To get familiar with processing of strings
- To get familiar with various operations that can be performed on strings through built-in functions
- Take examples related to string processing

Revision: 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'>
```

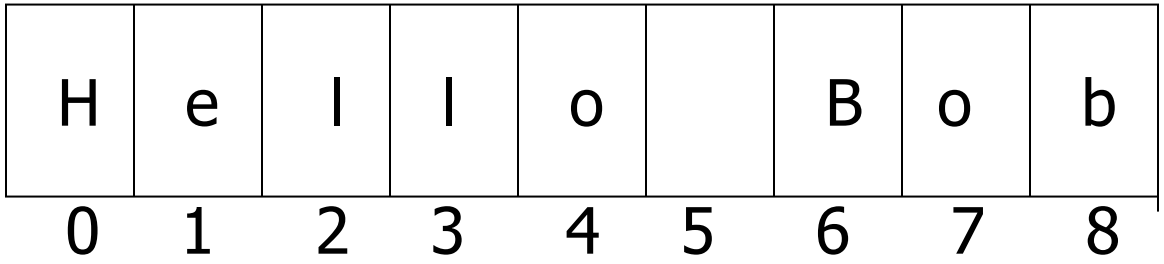
Revision: String indexing



In a string of *n* characters, the last character is at position *n-1* since we start counting with 0.

```
>>> greet[1]
'e'
>>> greet[6]
'B'
>>> greet[-1]
'b'
>>> greet[-3]
'B'
```

Revision: String slicing / substring




```
>>> greet[0:3]
'Hel'
>>> greet[5:9]
' Bob'
>>> greet[:5]
'Hello'
>>> greet[5:]
' Bob'
>>> greet[:]
'Hello Bob'
```

This is same as `greet`

Revision: String operations

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

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
- http://www.asciitable.com/

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'
```

Note that `'A' < 'a'`

Programming an Encoder

- 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>

Programming an Encoder

```
# 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:
        print(ord(ch),  end=",")
    print()  # Go to new line at the end of code sequence
```

Replace new line by comma

Programming an Encoder

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,

>>>
```

Programming an Encoder

```
# 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]))
```

Programming a Decoder

- 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
```

Programming a Decoder

- 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.

Programming a Decoder

- 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 → list of digit strings → list of numbers → string

`“102,114” → [“102”,”114”] → [102,114] → “fr”`

Programming a Decoder

- 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()
['Hello', 'string', 'methods!']
```

This is a list. More about them in the next lecture

- This is the same as:

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

Programming a Decoder

- 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']
```

Programming a Decoder

```
# numbers2text.py
#     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)
```

Programming a Decoder

- 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.*

Programming a Decoder

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.

More String Methods

- There are a number of other string methods. Try them all! https://docs.python.org/3/library/stdtypes.html#string-methods
  - `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*

More String Methods

- `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*

More String Methods

- `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 right-most position*

More String Methods

- `s.rstrip()` – *Copy of s with trailing whitespace removed*
- `s.split()` – *Split s 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 about functions which can be used with strings.
- We took examples where various operations can be performed on strings.
- We learned fundamental concept of Cryptography.