Politecnico
di Milano

Coding in Python

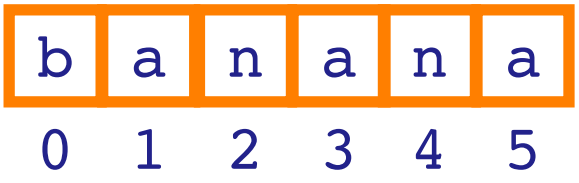
10-14 Giugno 2018

Lezione 2

STRINGS

Looking Inside Strings

- We can get at any single character in a string using an index specified in square brackets
- The index value must be an integer and starts at zero
- The index value can be an expression that is computed




```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

Looping Through Strings

- Using a while statement, an iteration variable, and the len function, we can construct a loop to look at each of the letters in a string individually

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

```
0 b
1 a
2 n
3 a
4 n
5 a
```

Looping Through Strings

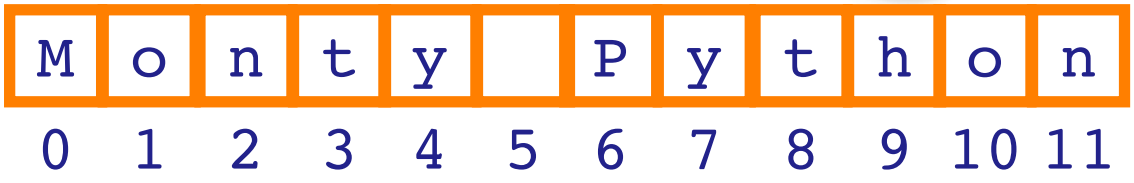
- A definite loop using a for statement is much more elegant
- The iteration variable is completely taken care of by the for loop

```
fruit = 'banana'
for letter in fruit:
    print(letter)
```

```
b
a
n
a
n
a
```

Slicing Strings

- We can also look at any continuous section of a string using a colon operator
- The second number is one beyond the end of the slice - “up to but not including”
- If the second number is beyond the end of the string, it stops at the end




```
>>> s = 'Monty Python'
>>> print(s[0:4])
'Mont'
>>> print(s[6:8])
'Py'
>>> print(s[:2])
'Mo'
>>> print(s[8:])
'thon'
>>> print(s[:])
'Monty Python'
>>> print(s[7:3:-1])
'yP y'
>>> print(s[::-1])
'nohtyP ytnoM'
```

Exercise

- Write a program that decides whether a word is a palindrome
  - adda
  - ottetto
  - radar

Solution 1


```
word = input('Insert a word ')
i = 0
j = len(word)-1
while i<j/2 and word[i]==word[j]:
    i = i+1
    j = j-1
if i >= j/2:
    print('the word is palindrome')
else :
    print('the word is not palindrome')
```

Solution 2


```
word = input('Insert a word ')
if word == word[::-1]:
    print('the word is palindrome')
else :
    print('the word is not palindrome')
```

String Concatenation

- When the + operator is applied to strings, it means "concatenation"

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print(b)
HelloThere
>>> c = a + ' ' + 'There'
>>> print(c)
Hello There
>>>
```

Using in as a Logical Operator

- The in keyword can also be used to check to see if one string is “in” another string
- The in expression is a logical expression that returns True or False and can be used in an if statement

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Found it!')
...
Found it!
>>>
```

String Library

- Python has a number of string functions which are in the string library
- These functions are already built into every string - we invoke them by appending the function to the string variable
- These functions do not modify the original string, instead they return a new string that has been altered

```
>>> greet = 'Hello Bob'
>>> zap = greet.lower()
>>> print(zap)
hello bob
>>> print(greet)
Hello Bob
>>> print('Hi There'.lower())
hi there
>>>
```

String Library

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

https://docs.python.org/3/library/stdtypes.html#string-methods

String Library

`str.`**`replace`**(*old*, *new*[, *count*])

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

`str.`**`rfind`**(*sub*[, *start*[, *end*]])

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within `s[start:end]`. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

`str.`**`rindex`**(*sub*[, *start*[, *end*]])

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

`str.`**`rjust`**(*width*[, *fillchar*])

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

`str.`**`rpartition`**(*sep*)

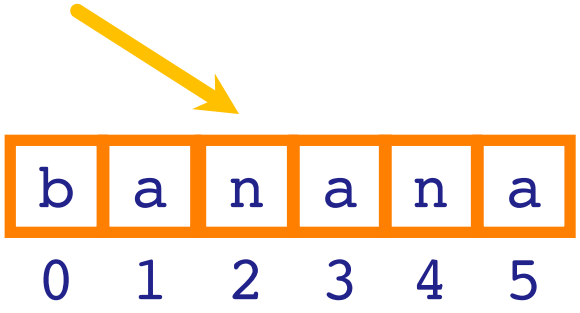
Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

`str.`**`rsplit`**(*sep=None*, *maxsplit=-1*)

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below.

Searching a String

- We use the find() function to search for a substring within another string
- find() finds the first occurrence of the substring
- If the substring is not found, find() returns -1
- Remember that string position starts at zero




```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

Making everything UPPER CASE

- You can make a copy of a string in lower case or upper case
- Often when we are searching for a string using find() we first convert the string to lower case so we can search a string regardless of case

```
>>> greet = 'Hello Bob'
>>> nnn = greet.upper()
>>> print(nnn)
HELLO BOB
>>> www = greet.lower()
>>> print(www)
hello bob
>>>
```

Search and Replace

- The replace() function is like a “search and replace” operation in a word processor
- It replaces all occurrences of the search string with the replacement string

```
>>> greet = 'Hello Bob'
>>> nstr = greet.replace('Bob','Jane')
>>> print(nstr)
Hello Jane
>>> nstr = greet.replace('o','X')
>>> print(nstr)
HellX BXb
>>>
```

Stripping Whitespace

- Sometimes we want to take a string and remove whitespace at the beginning and/or end
- lstrip() and rstrip() remove whitespace at the left or right
- strip() removes both beginning and ending whitespace

```
>>> greet = '   Hello Bob  '
>>> greet.lstrip()
'Hello Bob  '
>>> greet.rstrip()
'   Hello Bob'
>>> greet.strip()
'Hello Bob'
>>>
```

Prefixes


```
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

Parsing and extracting

```
                    21          31
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ',atpos)
>>> print(sppos)
31
>>> host = data[atpos+1 : sppos]
>>> print(host)
uct.ac.za
```

LISTS

A List is a Kind of Collection

- A collection allows us to put many values in a single "variable"
- A collection is nice because we can carry all many values around in one convenient package.

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]

carryon = [ 'socks', 'shirt', 'perfume' ]
```

List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object - even another list
- A list can be empty

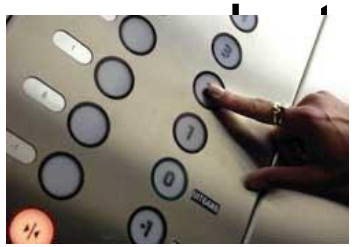
```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow', 'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([ 1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

A use of lists we have already seen


```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Blastoff!')
```

5
4
3
2
1
Blastoff!

Looking Inside Lists



like strings, we can get at any single element in using an index specified in square brackets

Joseph	Glenn	Sally
0	1	2

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> print(friends[1])
Glenn
>>>
```

How Long is a List?

- The len() function takes a list as a parameter and returns the number of elements in the list
- Actually len() tells us the number of elements of any set or sequence (such as a string...)

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
>>>
```

Using the range Function

- The range function returns a list of numbers that range from zero to one less than the parameter
- We can construct an index loop using for and an integer iterator

```
>>> print(range(4))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn',
'Sally']
>>> print(len(friends))
3
>>> print(range(len(friends)))
[0, 1, 2]
>>>
```

Concatenating Lists Using +

- We can create a new list by adding two existing lists together

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

Lists Can Be Sliced Using :


```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
>>> s = t[:]
>>> s
[9, 41, 12, 3, 74, 15]
```

Remember: the second number is “up to but not including”

List Methods


```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
['append', 'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
>>>
```

http://docs.python.org/tutorial/datastructures.html

Building a List from Scratch

- We can create an empty list and then add elements using the append method
- The list stays in order and new elements are added at the end of the list

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

Is Something in a List?

- Python provides two operators that let you check if an item is in a list
- These are logical operators that return True or False
- They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```

Comprehensions: basics

- It is a way of creating a new list from an existing one.
- Its syntax is derived from a construct in set theory notation that applies an operation to each item in a set
  - >>> L = [1,2,3,4,5]
  - >>> res = [x + 10 for x in L]
- is equivalent to
  - >>> res = [] # or res = list()
  - >>> for x in L :
  - … res.append(x+10)

[*expression* for *var* in *list*]

list comprehensions are introduced by square brackets
…we are creating a list…

Python executes an iteration across L

Comprehensions are typically a lot faster than using for loops explicitly

Example

- a = [chr(ord('a')+i) for i in range(26)]
- Equivalent to
  - a= []
  - for i in range(26)
  - a.append(chr(ord(‘a’)+i))

A numerical value corresponding to ‘a’

Strings are immutable and lists are mutable

- Strings are “immutable” - we cannot change the contents of a string - we must make a new string to make any change
- Lists are “mutable” - we can change an element of a list using the index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not
support item assignment
```

```
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

Python types and immutability

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

- From https://medium.com/@meghamohan/mutable-and-immutable-side-of-python-c2145cf72747

Built-in Functions and Lists

- There are a number of functions built into Python that take lists as parameters

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25.6
```

Best Friends: Strings and Lists


```
>>> abc = 'With three words'
>>> stuff = abc.split()
>>> print(stuff)
['With', 'three', 'words']
>>> print(len(stuff))
3
>>> print(stuff[0])
With
```

```
>>> print(stuff)
['With', 'three', 'words']
>>> for w in stuff :
...     print(w)
...
With
Three
Words
>>>
```

Split breaks a string into parts and produces a list of strings. We think of these as words. We can access a particular word or loop through all the words.

Best Friends: Strings and Lists


```
>>> line = 'A lot                of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print(thing)
['first;second;third']
>>> print(len(thing))
1
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
>>>
```

When you do not specify a delimiter, multiple spaces are treated like one delimiter

You can specify what delimiter character to use in the splitting

The Double Split Pattern

- Sometimes we split a line one way, and then grab one of the pieces of the line and split that piece again

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
```

The Double Split Pattern


```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

words = line.split()
email = words[1]                    stephen.marquard@uct.ac.za
print email
```

The Double Split Pattern


```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
pieces = email.split('@')
print pieces[1]
```

```
stephen.marquard@uct.ac.za
['stephen.marquard', 'uct.ac.za']
```

The Double Split Pattern

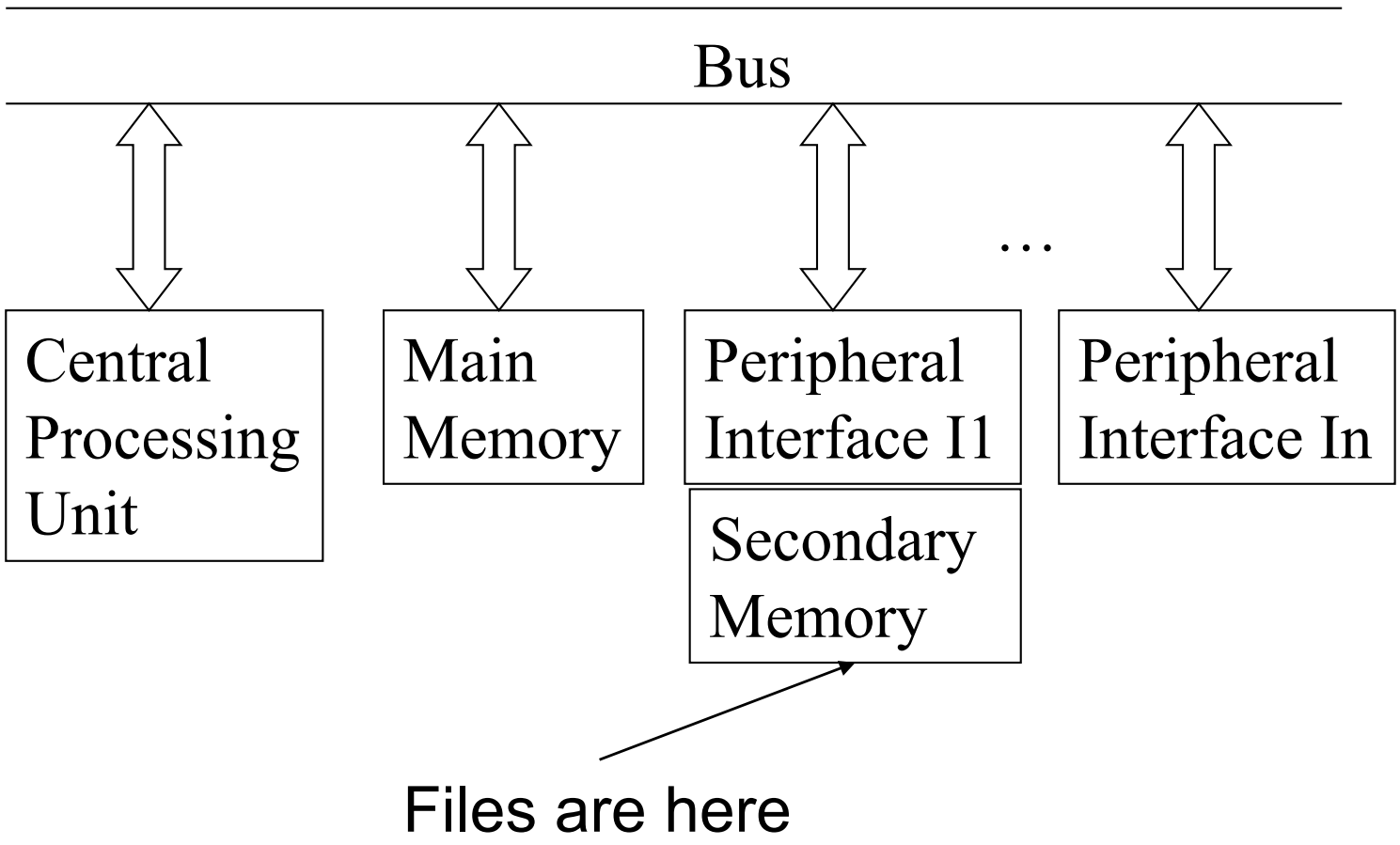

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
pieces = email.split('@')
print(pieces[1])
```

```
stephen.marquard@uct.ac.za
['stephen.marquard', 'uct.ac.za']
'uct.ac.za'
```

FILES

Where are files?



File Processing

- A text file can be thought of as a sequence of lines

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/

Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

http://www.py4e.com/code/mbox-short.txt

Opening a File

- Before we can read the contents of the file, we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the open() function
- open() returns a “file handle” - a variable used to perform operations on the file
- Similar to “File -> Open” in a Word Processor
- File handles are not numbers, sequences or mappings and they do not respond to expression operators

Common file operations

- >>> aFile = open(filename, mode)
- >>> aFile.method()

Operation	Interpretation
`output = open(r'C:\spam', 'w')`	Create output file ('w' means write)
`input = open('data', 'r')`	Create input file ('r' means read)
`input = open('data')`	Same as prior line ('r' is the default)
`aString = input.read()`	Read entire file into a single string
`aString = input.read(N)`	Read up to next N characters (or bytes) into a string
`aString = input.readline()`	Read next line (including \n newline) into a string
`aList = input.readlines()`	Read entire file into list of line strings (with \n)
`output.write(aString)`	Write a string of characters (or bytes) into file
`output.writelines(aList)`	Write all line strings in a list into file
`output.close()`	Manual close (done for you when file is collected)
`output.flush()`	Flush output buffer to disk without closing
`anyFile.seek(N)`	Change file position to offset N for next operation
`for line in open('data'): use line`	File iterators read line by line
`open('f.txt', encoding='latin-1')`	Python 3.X Unicode text files (str strings)
`open('f.bin', 'rb')`	Python 3.X bytes files (bytes strings)
`codecs.open('f.txt', encoding='utf8')`	Python 2.X Unicode text files (unicode strings)
`open('f.bin', 'rb')`	Python 2.X bytes files (str strings)

modes

read(r), write(w), append(a)
binary(b)
both input and output (+)

Using files

```
>>> myFile = open('myFile.txt', 'w')
>>> myFile.write('hello text file\n')
16
>>> myFile.write('goodbye text file\n')
18
>>> myFile.close()

>>> myFile.open('myFile.txt')
>>> myFile.readline()
'hello text file\n'
>>> myFile.readline()
'goodbye text file\n'
>>> myFile.readline()
''
```

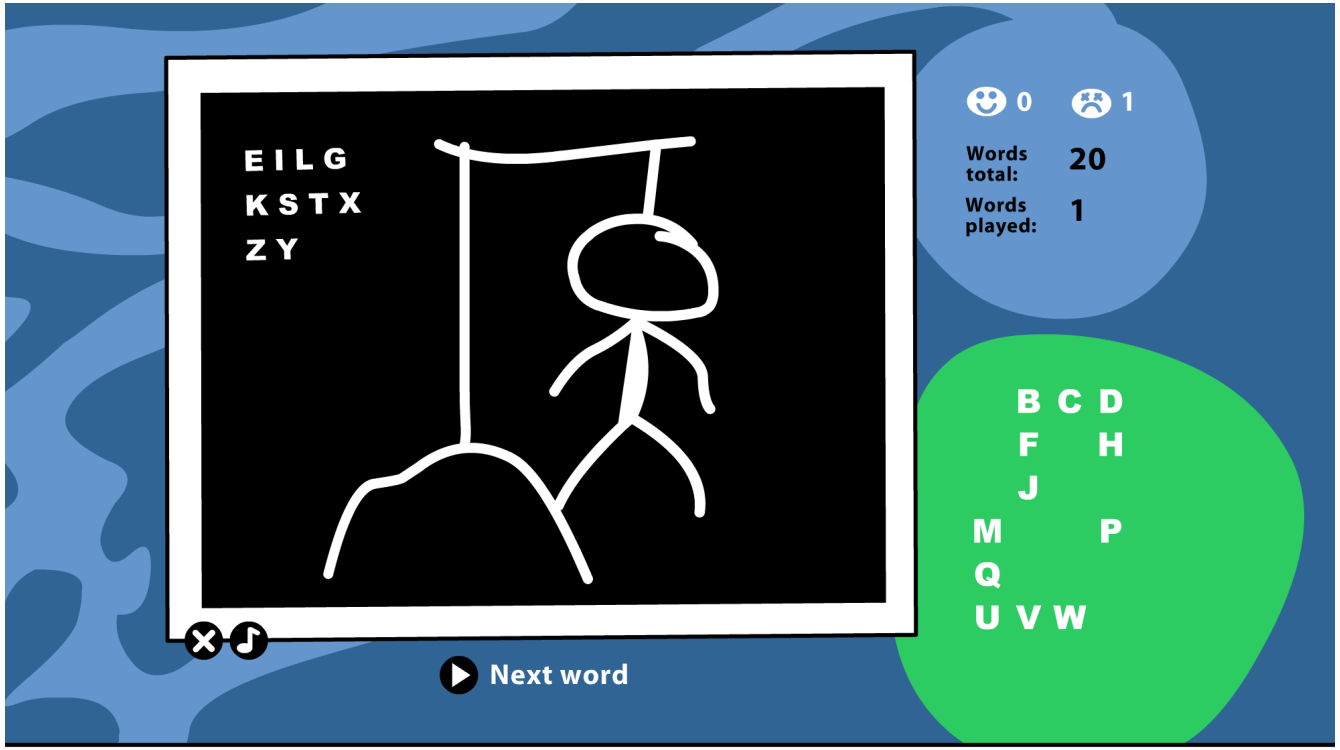
```
>>> for line in open('myFile.txt'):
        print(line)

hello text file
goodbye text file
```

Esercizio

- Scrivere un programma che legge un testo da file e conta il numero di articoli determinativi e indeterminativi nel file

Impiccato (Hangman)

- Sviluppare un programma per il gioco dell'impiccato



J O H N R A M B O

http://www.hangman.no

Altri esercizi

- Studiare la documentazione sulle operazioni per le stringhe e le liste e provare a utilizzare le operazioni che interessano di più
- Cercare su wikipedia il significato di file csv
- Scrivere un programma che, dato un file strutturato secondo il formato csv e contenente nome, cognome e voto di un insieme di studenti, calcola la media dei voti

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Adaptation and extensions for the Geo-Python Lab needs: Elisabetta Di Nitto, Politecnico di Milano

Other slides have been adapted from material by my colleague Prof. Sam Guinea, Politecnico di Milano