# Strings – Analysis of a text

*You're going to do some fun activities by manipulating strings and characters.*

**Lesson 1** (Characters and strings).

1. A *character* is a unique symbol, some examples of characters include: a lowercase letter `"a"`, a capital letter `"B"`, a special symbol `"&"`, a symbol representing a number `"7"`, a space `" "` that we will also note `"␣"`.
   To designate a character, it must be put in single quotation marks `'z'` or double quotation marks `"z"`.
2. A *string* is a sequence of characters, such as a word `"Hello"`, a sentence `'It is sunny.'` or a password `"N[w5ms}e!"`.
3. The type of a character or string is `str`.

**Lesson 2** (Operations on strings).

1. The *concatenation*, i.e. the end-to-end placing of two strings, is done using the operator `+`. For example `"umbr" + "ella"` gives the string `"umbrella"`.
2. The empty string `""` is useful when you want to initialize a string before adding other characters.
3. The *length* of a string is the number of characters it contains. It is obtained by calling the function `len()`. For example `len("Hello␣World")` returns 11 (a space counts as a character).
4. If `word` is a string then you can retrieve each character by the command `word[i]`. For example, if `word = "plane"` then:
   - `word[0]` is the character `"p"`,
   - `word[1]` is the character `"l"`,
   - `word[2]` is the character `"a"`,
   - `word[3]` is the character `"n"`,
   - `word[4]` is the character `"e"`.

| Letter | p | l | a | n | e |
|---|---|---|---|---|---|
| Rank | 0 | 1 | 2 | 3 | 4 |

Note that there are 5 letters in the word `"plane"` and that you access it through the ranks starting with 0. The indices are therefore 0, 1, 2, 3, and 4 for the last letter. More generally, if `word` is a string, characters are obtained by `word[i]` for `i` varying from 0 to `len(word)-1`.

**Lesson 3** (Substrings).
You can extract several characters from a string using the syntax `word[i:j]` which returns a string formed by characters ranked $i$ to $j-1$ (beware the character number $j$ is not included).
For example if `word = "wednesday"` then:

- `word[0:4]` returns the `"wedn"` substring formed by the characters of ranks 0, 1, 2 and 3 (but not 4),
- `word[3:6]` returns `"nes"` corresponding to indices 3, 4 and 5.

| Letter | w | e | d | n | e | s | d | a | y |
|---|---|---|---|---|---|---|---|---|---|
| Rank | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Another example: `word[1:len(word)-1]` returns the word but with its first and last letter cut off.

**Activity 1** (Plurals of words).

*Goal: write a step by step program that returns the plural of a given word.*

1. For a string `word`, for example `"cat"`, the program should display the plural of this word by adding an `"s"`.
2. For a word, for example `"bus"`, it should display the last letter of this string (here `"s"`). Improve your program for the first question, by testing if the last letter is already an `"s"`:
   - if this is the case, then add `"es"` to form the plural (`"bus"` becomes `"buses"`),
   - otherwise you have to add `"s"`.
3. Check if a word ends with a `"y"`. If so, display the plural with `"ies"` (the plural of `"city"` is `"cities"`). (Exceptions are not taken into account.)
4. Wrap all your work from the first three questions in a function called `plural()`. The function displays nothing, but returns the word in its plural form.

**`plural()`**

Use: `plural(word)`
Input: a word (a string)
Output: the plural of the word

Examples:
- `plural("cat")` returns `"cats"`
- `plural("bus")` returns `"buses"`
- `plural("city")` returns `"cities"`

5. Write a function `conjugation()` that conjugates a verb to the present continuous tense.

**`conjugation()`**

Use: `conjugation(verb)`
Input: a verb (a string, exceptions are not taken into account)
Output: no result but displays conjugation in the present continuous tense

Example: `conjugation("sing")`, prints `"I am singing, you are singing,..."`

**Lesson 4** (A little more on strings).

1. A `for ... in ...` loop allows you to browse a string, character by character:

```
for charac in word:
    print(charac)
```

2. You can test if a character belongs to a certain list of characters. For example:

```
if charac in ["a", "A", "b", "B", "c", "C"]:
```

allows you to execute instructions if the character `charac` is one of the letters a, A, b, B, c, C.
To avoid some letters, we would use:

```
if charac not in ["X", "Y", "Z"]:
```

**Activity 2** (Word games).

*Goal: manipulating words in a fun way.*

1. **Distance between two words.**
The Hamming distance between two words of the same length is the number of places where the letters are different.
For example:

**SNAKE** **STACK**

The second letter of **SNAKE** is different from the second letter of **STACK**, the fourth and fifth ones are also different. The Hamming distance between **SNAKE** and **STACK** is therefore equal to 3.
Write a function `hamming_distance()` that calculates the Hamming distance between two words of the same length.

`hamming_distance()`

Use: `hamming_distance(word1,word2)`
Input: two words (strings)
Output: the Hamming distance (an integer)

Example: `hamming_distance("SHORT","SKIRT")` returns 2

2. **Upside down.**
Write a function `upsidedown()` that returns a word backwards: **HELLO** becomes **OLLEH**.

`upsidedown()`

Use: `upsidedown(word)`
Input: a word (a string)
Output: the word backwards

Example: `upsidedown("PYTHON")` returns `"NOHTYP"`

3. **Palindrome.**
Deduce a function that tests whether a word is a palindrome or not. A *palindrome* is a word that can be written from left to right or right to left; for example **RADAR** is a palindrome.

**is_palindrome()**

Use: `is_palindrome(word)`
Input: a word (a string)
Output: “True” if the word is a palindrome, “False” otherwise.

Example: `is_palindrome("KAYAK")` returns `True`

4. **Pig latin.** Pig latin is a made up language, here are the rules according to *Wikipedia*:
   - For words that begin with vowel sounds, one just adds "way" to the end. Examples are:
     - **EAT** becomes **EATWAY**
     - **OMELET** becomes **OMELETWAY**
     - **EGG** becomes **EGGWAY**
   - For words that begin with consonant sounds, all letters before the initial vowel are placed at the end of the word sequence. Then, "ay" is added, as in the following examples:
     - **PIG** becomes **IGPAY**
     - **LATIN** becomes **ATINLAY**
     - **BANANA** becomes **ANANABAY**
     - **STUPID** becomes **UPIDSTAY**
     - **GLOVE** becomes **OVEGLAY**

   Write a function `pig_latin()` that translates into pig latin a word according to this procedure.

**pig_latin()**

Use: `pig_latin(word)`
Input: a word (a string)
Output: the word transformed into pig latin.

Examples:
- `pig_latin("DUCK")` returns `"UCKDAY"`
- `pig_latin("ALWAYS")` returns `"ALWAYSWAY"`

**Activity 3** (DNA)**.**

*A DNA molecule is made up of about six billion nucleotides. The computer is therefore an essential tool for DNA analysis. In a DNA strand there are only four types of nucleotides that are noted **A**, **C**, **T** or **G**. A DNA sequence is therefore a long word in the form: **TAATTACAGACACCTGAA...***

1. Write a function `presence_of_A()` that tests if a sequence contains the nucleotide **A**.

**presence_of_A()**

Use: `presence_of_A(sequence)`
Input: a DNA sequence (a string whose characters are among A, C, T, G)
Output: “True” if the sequence contains “A”, “False” otherwise.

Example: `presence_of_A("CTTGCT")` returns `False`

2. Write a function `position_of_AT()` that tests if a sequence contains the nucleotide **A** followed by the nucleotide **T** and returns the position of the first occurrence found.

**position_of_AT()**

Use: `position_of_AT(sequence)`
Input: a DNA sequence (a string whose characters are among A, C, T, G)
Output: the position of the first "AT" sequence found (starting at 0); `None` if not found.

Example:

- `position_of_AT("CTTATGCT")` returns 3
- `position_of_AT("GATATAT")` returns 1
- `position_of_AT("GACCGTA")` returns `None`

*Hint.* `None` is assigned to a variable to indicate the absence of a value.

3. Write a function `position()` that tests if a sequence contains a given code and returns the position of the first occurrence.

**position()**

Use: `position(code,sequence)`
Input: a code and a DNA sequence
Output: the position of the beginning of the code found; `None` if not found

Example: `position("CCG","CTCCGTT")` returns 2

4. A crime has been committed in the castle of Adeno. You recovered two strands of the culprit's DNA, from two distant positions in the DNA sequence. There are four suspects, whose DNA you sequenced. Can you find out who did it?
First code of the culprit: **CATA**
Second code of the culprit: **ATGC**

DNA of Colonel Mustard:
**CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC**
Miss Scarlet's DNA:
**CTCCTGATGCTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG**
Mrs. Peacock's DNA:
**AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGTACTCCGCGCGCCGGGACAGAATGCC**
Pr. Plum's DNA:
**CTGCAGGAACTTCTTCTGGAAGTACTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG**

**Lesson 5** (Character encoding).
A character is stored by the computer as an integer. For ASCII/unicode encoding, the capital letter "A" is encoded by 65, the lowercase letter "h" is encoded by 104, and the symbol "#" by 35.
Here is the table of the first characters. Numbers 0 to 32 are not printable characters. However, the number 32 is the space character "␣".

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 33 | ! | 43 | + | 53 | 5 | 63 | ? | 73 | I | 83 | S | 93 | ] | 103 | g | 113 | q | 123 | { |
| 34 | " | 44 | , | 54 | 6 | 64 | @ | 74 | J | 84 | T | 94 | ^ | 104 | h | 114 | r | 124 | \| |
| 35 | # | 45 | - | 55 | 7 | 65 | A | 75 | K | 85 | U | 95 | _ | 105 | i | 115 | s | 125 | } |
| 36 | $ | 46 | . | 56 | 8 | 66 | B | 76 | L | 86 | V | 96 | ‘ | 106 | j | 116 | t | 126 | ~ |
| 37 | % | 47 | / | 57 | 9 | 67 | C | 77 | M | 87 | W | 97 | a | 107 | k | 117 | u | 127 | - |
| 38 | & | 48 | 0 | 58 | : | 68 | D | 78 | N | 88 | X | 98 | b | 108 | l | 118 | v | | |
| 39 | ’ | 49 | 1 | 59 | ; | 69 | E | 79 | O | 89 | Y | 99 | c | 109 | m | 119 | w | | |
| 40 | ( | 50 | 2 | 60 | < | 70 | F | 80 | P | 90 | Z | 100 | d | 110 | n | 120 | x | | |
| 41 | ) | 51 | 3 | 61 | = | 71 | G | 81 | Q | 91 | [ | 101 | e | 111 | o | 121 | y | | |
| 42 | * | 52 | 4 | 62 | > | 72 | H | 82 | R | 92 | \ | 102 | f | 112 | p | 122 | z | | |

1. The function `chr()` is a `Python` function that returns the character associated to a code.

**python : chr()**

Use: `chr(code)`
Input: a code (an integer)
Output: a character

Example:
- `chr(65)` returns `"A"`
- `chr(123)` returns `"{"`

2. The function `ord()` is a `Python` function corresponding to the reverse operation: it returns the code associated with a character.

**python : ord()**

Use: `ord(charac)`
Input: a character (a string of length 1)
Output: an integer

Example:
- `ord("A")` returns `65`
- `ord("*")` returns `42`

**Activity 4** (Upper case/lower case).

*Goal: convert a word to upper or lower case.*

1. Decode by hand the encrypted hidden message:
   80-121-116-104-111-110 105-115 99-64-64-108
2. Write a loop that displays characters encoded by integers from 33 to 127.
3. What does the command `chr(ord("a")-32)` return? And `chr(ord("B")+32)`?

4. Write a function `upper_letter()` that transforms a lowercase letter into its uppercase letter.

**`upper_letter()`**

Use: `upper_letter(charac)`
Input: a lowercase character among `"a"`,...,`"z"`
Output: the same letter in upper case

Example: `upper_letter("t")` returns `"T"`

5. Write a function `uppercase()` that takes a sentence written in lower case and returns the same sentence written in upper case. Characters that are not lowercase letters remain unchanged.

**`uppercase()`**

Use: `uppercase(sentence)`
Input: a sentence
Output: the same sentence in upper case

Example: `uppercase("Hello world!")` returns `"HELLO WORLD!"`

Do the same thing for a `lowercase()` function.

6. Write a function `format_full_name()` that returns the first name and last name formatted according to the style **First_name LAST_NAME**.

**`format_full_name()`**

Use: `format_full_name(somebody)`
Input: a person's first name and surname (separated by a space)
Output: the full name following the format `"First name LAST NAME"`

Example:
- `format_full_name("harry Potter")` returns `"Harry POTTER"`
- `format_full_name("LORD Voldemort")` returns `"Lord VOLDEMORT"`

**Activity 5.**

*Goal: determine the language of a text from the analysis of letter frequencies.*

1. Write a function called `occurrences_letter()` that counts the number of times the given letter appears in a sentence (in upper case).

**`occurrences_letter()`**

Use: `occurrences_letter(letter,sentence)`
Input: a letter (a character) and a sentence in capital letters (a string)
Output: the number of occurrences of the letter (an integer)

Example: `occurrences_letter("E","IS THERE ANYBODY OUT THERE")` returns 4

2. Write a function called `number_letters()` that counts the total number of letters that appear in a sentence (in upper case). Do not count spaces or punctuation.

**`number_letters()`**

Use: `number_letters(sentence)`
Input: a sentence in capital letters (a string)
Output: the total number of letters from "A" to "Z" (an integer)

Example: `number_letters("IS THERE ANYBODY OUT THERE")` returns `22`.

3. The *frequency of appearance* of a letter in a text or sentence is the percentage given according to the formula :

$$\text{frequency of appearance of a letter} = \frac{\text{number of occurrences of the letter}}{\text{total number of letters}} \times 100.$$

For example, the sentence **IS THERE ANYBODY OUT THERE** contains 22 letters; the letter **E** appears there 4 times. The frequency of appearance of **E** in this sentence is therefore:

$$f_E = \frac{\text{number of occurrences of E}}{\text{total number of letters}} \times 100 = \frac{4}{22} \times 100 \simeq 16.66$$

The frequency is therefore about 17%.
Write a function called `percentage_letter()` that calculates this frequency of appearance.

**`percentage_letter()`**

Use: `percentage_letter(letter,sentence)`
Input: a letter (a character) and a sentence in capital letters (a string)
Output: the frequency of appearance of the letter (a number lower than 100)

Example: `percentage_letter("E","IS THERE ANYBODY OUT THERE")` returns `16.66...`

Use this function to properly display the frequency of appearance of all letters in a sentence.

4. Here is the frequency of appearance of letters according to the language used (source: en.wikipedia.org/wiki/Letter_frequency). For example, the most common letter in English is "E" with a frequency of more than 12%. The "W" represents about 2% of letters in English and German, but almost does not appear in French and Spanish. These frequencies also vary according to the text analyzed.

| Letter | English | French | German | Spanish |
|---|---|---|---|---|
| a | 8.167% | 8.173% | 7.094% | 12.027% |
| b | 1.492% | 0.901% | 1.886% | 2.215% |
| c | 2.782% | 3.345% | 2.732% | 4.019% |
| d | 4.253% | 3.669% | 5.076% | 5.010% |
| e | 12.702% | 16.734% | 16.396% | 12.614% |
| f | 2.228% | 1.066% | 1.656% | 0.692% |
| g | 2.015% | 0.866% | 3.009% | 1.768% |
| h | 6.094% | 0.737% | 4.577% | 0.703% |
| i | 6.966% | 7.579% | 6.550% | 6.972% |
| j | 0.153% | 0.613% | 0.268% | 0.493% |
| k | 0.772% | 0.049% | 1.417% | 0.011% |
| l | 4.025% | 5.456% | 3.437% | 4.967% |
| m | 2.406% | 2.968% | 2.534% | 3.157% |
| n | 6.749% | 7.095% | 9.776% | 7.023% |
| o | 7.507% | 5.819% | 3.037% | 9.510% |
| p | 1.929% | 2.521% | 0.670% | 2.510% |
| q | 0.095% | 1.362% | 0.018% | 0.877% |
| r | 5.987% | 6.693% | 7.003% | 6.871% |
| s | 6.327% | 7.948% | 7.577% | 7.977% |
| t | 9.056% | 7.244% | 6.154% | 4.632% |
| u | 2.758% | 6.429% | 5.161% | 3.107% |
| v | 0.978% | 1.838% | 0.846% | 1.138% |
| w | 2.360% | 0.074% | 1.921% | 0.017% |
| x | 0.150% | 0.427% | 0.034% | 0.215% |
| y | 1.974% | 0.128% | 0.039% | 1.008% |
| z | 0.074% | 0.326% | 1.134% | 0.467% |

In your opinion, which languages were written the following four texts written in (the letters of each word were mixed)?

TMAIER BERACUO RSU NU REBRA PRCEEH EIANTT NE ONS EBC NU GAOFREM EIMATR RERNAD APR L RDUOE LAHECLE UIL TTNI A EUP SREP EC LGNGAEA TE RBONUJO ERMNOUSI DU UBRACEO QUE OVSU EEST LIJO UQE OUVS EM MSZELBE BAEU ASNS MIERNT IS RVETO AGRAME ES PRARPTOE A OEVTR AMGUPLE VUOS SEET EL PNIHXE DSE OSHET ED CSE BIOS A ESC MSOT LE OUBRCEA NE ES ESTN ASP DE IEJO TE OUPR ERRNOTM AS BELEL XOVI IL OREVU NU RGLEA ECB ILESSA EBOMTR AS PIOER EL NRDAER S EN ISIAST TE ITD MNO NOB EUSRMNOI NRPEEAZP QEU UTOT EUTLRFTA IVT XUA SPNEDE DE UECIL UQI L TECEOU TECET NEOCL VATU BNEI UN GMAEORF SNAS TUOED LE EOABURC OHENTXU TE NSCOFU UJRA SMIA UN EPU TRDA UQ NO EN L Y ARRPEIDNT ULSP

WRE TREITE SO TSPA CUDHR AHNCT UND WIND SE STI RED AEVRT MTI ESEIMN IDNK RE ATH END NEABNK WLOH IN EMD AMR ER AFTSS HIN IHSERC RE AHTL HIN MRWA EINM SHNO SAW SRTIBG UD SO NGBA DNEI EIHSGTC ESISTH RAETV UD DEN LERNIOKG NITHC NDE LOENINKGRE TIM OKRN UDN CHWFSEI NEIM NSOH ES STI IEN BIFTRLSEEEN DU BILESE IKDN OMKM EHG MIT MIR RAG ECHNOS EPELSI EIPSL IHC ITM RDI HNCMA BEUTN MBLUNE DINS NA DEM TNDRAS NMIEE UTETMR AHT CAMHN UDNGEL GDAWEN MIEN EATRV MENI VEART DUN OSTHER DU CINTH SAW KNNOEIREGL RIM ILEES PRSTVRCIEH ISE IHGRU BEEILB RIGUH MNEI KNDI NI RDNEUR NATBRLET STAESUL EDR WNID

DSNOACAIF ORP ANU DAEDALRI DNAAEIMTI EQU NNCOSETE EL RSTEOUL SMA AACTFAITNS UQE EL TSVAO OINSRVUE DE US ANIGIICANOM EIORDP TOOD RTEIENS RPO LE ITOABOLRROA ED QIUAMALI USOP A NSSRCAEAD LA TMREAAI NXTADAUEE ROP GOARLS EMESS DE NNAMICLUIAPO Y LOVOIV A

> RES LE RHMEOB EOMDNEERPRD DE LOS RSOPMRIE OMTSIPE UEQ CIIDADE LE RTDAAOZ ED LSA CELSAL Y LA NICOIOPS ED LAS UESVNA SSACA Y ES ITRMNEEOD QEU AERFU EL UEQIN IIIRDEGAR LA NAIORTREICP DE AL RRTEIA
>
> IMTRUESMME DNA TEH LNGIIV SI EYAS SIFH REA GJPNUIM DNA HET TTNOCO IS GHIH OH OUYR DDADY SI IRHC DAN ROUY MA SI DOGO GKOILON OS USHH LTLIET BBYA NDOT OUY CYR NEO OF HESET GNSRONIM YUO RE NANGO SIER PU SNIGING NAD OULLY EPADRS YUOR GINSW DAN LYOLU KATE OT HET KSY TUB ITLL TATH MGNIRNO EREHT NATI INTGOHN ACN AHMR OYU TWIH DADYD NDA MYMMA NSTIDGAN YB