Lesson 4 – Data structures:

Strings and tuples

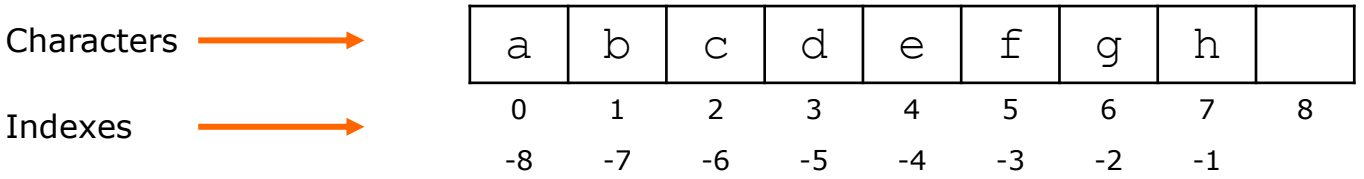
Remember: what is a string?

- A *str* is a string of characters.

```
In [55]: s = "abcdefgh"

In [56]: print(s)
abcdefgh
```

- How is it saved?



- Operator [] to access to the character at position i (index i-1)

```
In [58]: s[0]
Out[58]: 'a'
```

```
In [57]: s[2]
Out[57]: 'c'
```

```
In [62]: s[-1]
Out[62]: 'h'
```

```
In [63]: s[-8]
Out[63]: 'a'
```

- If we access a position outside the limits:

```
In [59]: s[8]
Traceback (most recent call last):

  File "<ipython-input-59-5cfd660ba969>", line 1, in <module>
    s[8]

IndexError: string index out of range
```

More about strings...

- Comparison strings (==, >, <, ...)

```
In [64]: "abc"=="abc"
Out[64]: True

In [66]: "abc"<="abd"
Out[66]: True

In [67]: "abc"!="abd"
Out[67]: True

In [68]: "abc">"abd"
Out[68]: False
```

- `len()` is a function that gives the length of the string

```
In [69]: s = "abcdefgh"

In [70]: len(s)
Out[70]: 8
```

- Operator `[start:stop:step]` returns the part of the string from the start character (included) to the stop (not included) taking each step (default step = 1)

```
In [71]: s[3:6]
Out[71]: 'def'

In [72]: s[3:6:2]
Out[72]: 'df'

In [73]: s[::]
Out[73]: 'abcdefgh'

In [74]: s[::-1]
Out[74]: 'hgfedcba'

In [75]: s[4:1:-2]
Out[75]: 'ec'

In [76]: s[:3]
Out[76]: 'abc'
```

More about strings...

- A string is immutable:

```
In [77]: s = "hello"

In [78]: s[0]
Out[78]: 'h'

In [79]: s[0] = "y"
Traceback (most recent call last):

  File "<ipython-input-79-bc25d351014d>", line 1, in <module>
    s[0] = "y"

TypeError: 'str' object does not support item assignment
```

immutable

- Alternative:

```
In [80]: s = "y" + s[1:len(s)]

In [81]: print(s)
yello
```

Example - Hashtags

Make a program that asks the user to enter a tweet (a string of characters) on the keyboard. The program must extract all hashtags.

E.g. "Hello #goodbye #IP" the program must print

#goodbye

#IP

Example – Hashtags (option 1)

Input 1→ Hello #hi and #bye friend

Input 2→ Hello #goodbye #world

Version 1

```
s=input("Write a tweet:")
long = len(s)

for i in range(0,long,1):
    if(s[i]=='#'):
        j=i+1
        while(s[j]!=" "):
            j=j+1
        print(s[i:j])
```

Input 1 ✔

Input 2 ✘

IndexError:
string index out of range

Version 2

```
s=input("Write a tweet:")
long = len(s)

for i in range(0,long,1):
    if(s[i]=='#'):
        j=i+1
        while(s[j]!=" ") and(j<long):
            j=j+1
        print(s[i:j])
```

Input 1 ✔

Input 2 ✘

IndexError:
string index out of range

Version 3

```
s=input("Write a tweet:")
long = len(s)

for i in range(0,long,1):
    if(s[i]=='#'):
        j=i+1
        while(j<long) and(s[j]!=" "):
            j=j+1
        print(s[i:j])
```

Input 1 ✔

Input 2 ✔

The order in conditionals matters!

Example – Hashtags (option 2)

Input 1→ Hello #hi and #bye friend

Input 2→ Hello #goodbye #world

```
s=input("Write a tweet:")
i=0
while(i<len(s)):
    if(s[i]=='#'):
        ini=i
        while(i<len(s)) and (s[i]!=" "):
            i+=1
        print(s[ini:i])
    i+=1
```

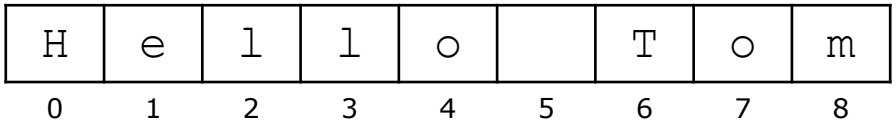
Input 1 ✔

Input 2 ✔

This code is more efficient than the previous one (option 1)

Tuple

- Like strings, tuples are ordered sequences of elements.



- The difference is that the elements of a tuple do not need to be characters.
- Tuple-type literals are written as a list of comma-separated items in parentheses.

```
In [15]: edats =(53, 44, 18, 16)

In [16]: edats
Out[16]: (53, 44, 18, 16)
```

```
In [70]: tupla = (1,)

In [71]: type (tupla)
Out[71]: tuple
```

```
In [72]: tupla = (1)

In [73]: type (tupla)
Out[73]: int
```

- Individual elements can be of any kind (heterogeneous structure) and do not need to be of the same type to each other.

```
In [17]: tupla =(53, "string", 3.5, 67)

In [18]: tupla
Out[18]: (53, 'string', 3.5, 67)
```

Indexing, cutting, concatenating

- We can access items via the position index, with the operator []:

```
In [38]: tupla[2]
Out[38]: 3.5

In [39]: tupla[0]
Out[39]: 53
```

- Operator [n:m] returns the sub-tuple from the n-th character (included) to m-th (not included):

```
In [40]: tupla[1:4]
Out[40]: ('string', 3.5, 67)

In [41]: tupla[1:2]
Out[41]: ('string',)
```

One extra coma means a tuple with 1 element

- We can concatenate two or more tuples:

```
In [42]: tupla + (99, "UAB")
Out[42]: (53, 'string', 3.5, 67, 99, 'UAB')
```

Immutable

- Tuples have fixed size:

```
In [74]: tupla =(53, "string", 3.5, 67)

In [75]: len(tupla)
Out[75]: 4
```

- Tuples are immutable: we cannot change values.

```
In [42]: tupla + (99, "UAB")
Out[42]: (53, 'string', 3.5, 67, 99, 'UAB')

In [43]: tupla[1] = "UAB"
Traceback (most recent call last):

  File "<ipython-input-43-165986f9bafb>", line 1, in <module>
    tupla[1] = "UAB"

TypeError: 'tuple' object does not support item assignment
```

Nested

- Tuples, like many of the data structures, can be nested:
  - we can form tuples of tuples.

```
In [77]: tupla = ("e0", "e1" , (1,2,3), "e3")

In [78]: len(tupla)
Out[78]: 4

In [79]: tupla[2]
Out[79]: (1, 2, 3)

In [80]: tupla[1:4]
Out[80]: ('e1', (1, 2, 3), 'e3')
```

Utilitity of tuples

- Tuples are really useful in a couple of different scenarios.
- **Swap**: Swap the value of two variables

```
In [44]: x=3

In [45]: y=5

In [46]: x=y

In [47]: y=x

In [48]: print(x,y)
5 5
```

```
In [49]: x=3

In [50]: y=5

In [51]: temp=x

In [52]: x=y

In [53]: y=temp

In [54]: print (x,y)
5 3
```

```
In [55]: x=3

In [56]: y=5

In [57]: (x,y)=(y,x)

In [58]: print(x,y)
5 3
```

- **To Return more than one value from a function*:**

```
def do_division(D,d):
    # D: dividend, d: divisor
    q = D // d   # quotient
    r = D % d    # remainder
    return (q,r)
```

```
In [3]: (quotient,remainder) = do_division(10,3)

In [4]: print(quotient,remainder)
3 1
```


* Functions will be seen later

Example

Make a program where at the beginning, a tuple is created that contains all the months of the year

(January, February, March, April, May, June, July, August, September, October, November, December).

The program must ask the user to enter a date in DD/MM/YYYY format (day, month, year separated by '/'). It should then display the date entered in the format:

"NAME_MONTH DD, YYYY"

where DD and YYYY are the day and year of the date entered by the user, and NAME_MONTH is the full name of the month number

```
mymonths = ("January", "February", "March", "April", "May", "June",
"July", "August", "September", "October", "November", "December")

#example: if 20/04/2025, then, m=4

print(mymonths[m-1])
```