PROGRAMMING 1 - WEEK 3

Strings

UC Leuven Limburg
MOVING MINDS

Recap

- Arithmetic

- Booleans
- Conditionals
- None

Permanent Evaluation

- Be quiet - no talking
- Don't cheat
  - Look at your own screen
- ONLY allowed to be on ANS
  - No GenAI – No VSCode – ...
  - No ppt – No notes – ...
- Browser full screen
- Display Light 100%
- Guess correction: -0.5 / +1
- Close your laptop when finished

PROGRAMMING 1 - WEEK 3

Strings

Data types

- Dealing with numbers
  - int
  - float

- Dealing with True/False
  - boolean

- Dealing with text
  - string

Strings – The basics

- Single or double quotes

```
message1 = "This is a valid string"
message2 = 'This is also a valid string'
```

Strings – The basics

- Single or double quotes
- Case sensitivity

```
message = "Strings are Case-Sensitive!"
different_message = "strings are CASE-sensitive!"
```

Strings – The basics

- Single or double quotes
- Case sensitivity
- Interpolation

```
name = "Jef"
print( f"My name is {name}" )
```

```
price = 100
discount = 20
price_after_discount = 100 * (1-discount/100)
print(f"The original price is €{price}, given a {discount}% discount, we only need to pay €{price_after_discount}.")
```

Terminal:

```
The original price is €100, given a 20% discount, we only need to pay €80.0.
```

Strings – The basics

- Single or double quotes
- Case sensitivity
- Interpolation
- Formatting

```
>>> h = 1
>>> m = 2
>>> s = 3
>>> f'{h:02}:{m:02}:{s:02}'
"01:02:03"
```

```
>>> pi = 3.141592
>>> f'{pi:.2f}'
"3.14"
```

```
>>> text = 'abc'
>>> f"{text:<10}|{text:>10}|{text:^10}|"
'abc       |       abc|   abc    |'
```

Strings – The basics

- Single or double quotes
- Case sensitivity
- Interpolation
- Formatting
- Operators

Syntax	Result
`"ab" + "cd"`	`"abcd"`
`"ab" * 3`	`"ababab"`
`"a" in "abc"`	`True`
`"a" == "a"`	`True`
`"a" != "a"`	`False`
`"a" < "b"`	`True`
`"a" <= "b"`	`True`
`"a" > "b"`	`False`
`"a" >= "b"`	`False`

```
"age" + 40

40 + "years"
```

```
"aa" == "AA"

"A" in "Name"
```

Unicode

ASCII Digits	U+0030	0
	U+0031	1
	U+0032	2
	U+0033	3
	U+0034	4
	U+0035	5
	U+0036	6
	U+0037	7
	U+0038	8
	U+0039	9

Latin Alphabet: Uppercase	U+0041	A
	U+0042	B
	U+0043	C
	U+0044	D
	U+0045	E
	U+0046	F
	U+0047	G
	U+0048	H
	U+0049	I
	U+004A	J
	U+004B	K
	U+004C	L
	U+004D	M
	U+004E	N
	U+004F	O
	U+0050	P
	U+0051	Q
	U+0052	R
	U+0053	S
	U+0054	T
	U+0055	U
	U+0056	V
	U+0057	W
	U+0058	X
	U+0059	Y
	U+005A	Z

ASCII Punctuation & Symbols	U+005B	[
	U+005C	\
	U+005D	]
	U+005E	^
	U+005F	_
	U+0060	`

Latin Alphabet: Lowercase	U+0061	a
	U+0062	b
	U+0063	c
	U+0064	d
	U+0065	e
	U+0066	f
	U+0067	g
	U+0068	h
	U+0069	i
	U+006A	j
	U+006B	k
	U+006C	l
	U+006D	m
	U+006E	n
	U+006F	o
	U+0070	p
	U+0071	q
	U+0072	r
	U+0073	s
	U+0074	t
	U+0075	u
	U+0076	v
	U+0077	w
	U+0078	x
	U+0079	y
	U+007A	z

Strings – The basics

- Single or double quotes
- Case sensitivity
- Interpolation
- Formatting
- Operators
- Escape Characters

```
"he said "hello" "
```

```
'he said "hello" '

"he said \"hello\" "
```

```
"You can use 'single' or \"double\" quotes"

'You can use \'single\' or "double" quotes'
```

Strings – The basics

- Single or double quotes
- Case sensitivity
- Interpolation
- Formatting
- Operators
- Escape Characters

Combination	Meaning
\"	"
\'	'
\n	Newline
\r	Carriage return
\t	Tab
\b	Backspace
\\	\

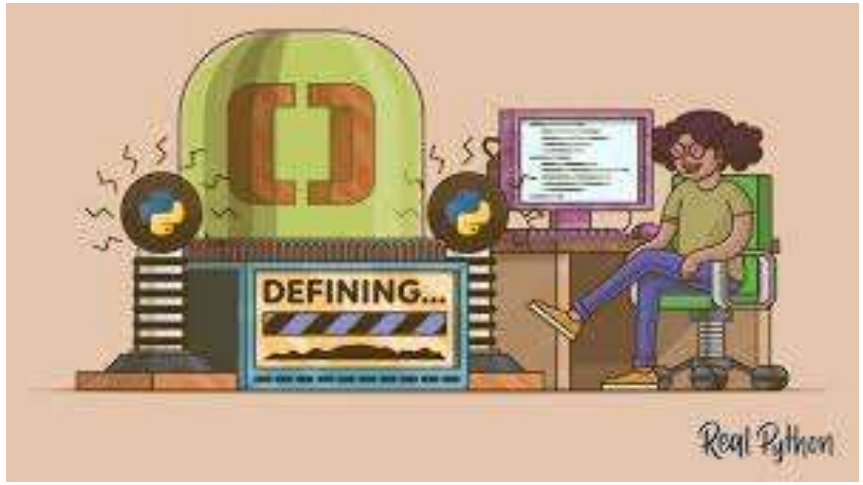
Strings – The basics

- Single or double quotes
- Case sensitivity
- Interpolation
- Formatting
- Operators
- Escape Characters
- Documentation

```
""" this is text that will be ignored by python"""

# this as well

''' this also '''

"""
this will
also be
ignored
"""

''' this
also '''
```

Strings – The basics

- Single or double quotes
- Case sensitivity
- Interpolation
- Formatting
- Operators
- Escape Characters
- Documentation
- Functions



Strings – Functions

- print() → output data to the terminal

```
print("Hello!")

print(1)
```

Strings – Functions

- print()
- input() → receive data from the terminal
  → data is captured as a string

```
name = input("Enter your name: ")

print(f"Hello {name}!")
```

```
age = input("Enter your age: ")

print( age + 1 )
```

Strings – Functions

- print()
- input()
- int()

```
age = int(input("Enter your age: "))
print( age + 1 )
```

```
average =(3+1)/2
print(average)
print(int(average))
print(average)
```

```
name = "Jef"
print(int(name))
```

Strings – Functions

- print()
- input()
- int()
- float()

```
score = input("Enter the score: ")
double_score = float(score) * 2
```

Strings – Functions

- print()
- input()
- int()
- float()
- str()

```
age = 20
print(age)
print("Age: " + str(age))
```

```
print(f"Age: {age}")
```

Strings – Functions

- print()
- input()
- int()
- float()
- str()
- len()

```
greeting = "Hello Jef!"
print(len(greeting))
```

Strings – The real deal

- Indexing
- Slicing
- Methods

Strings - Indexing

A string can be seen as a sequence of characters. You can access each of these characters individually.

```
>>> string = "abc"
>>> string[0]
"a"

>>> string[1]
"b"

>>> string[2]
"c"
```

```
>>> string = "abcd"
>>> string[-1]
"d"

>>> string[-2]
"c"
```

Zero based → First element has index zero!

Strings - Slicing

```
string = "abcdef"
first_three = string[0:3]
```

from index 0 (included)

to index 3 (excluded)

Other possibilities:

Syntax	Description
`s[:j]`	Same as `s[0:j]`
`s[i:]`	Same as `s[i:len(s)]`
`s[:]`	Same as `s[0:len(s)]`, i.e., returns the whole string

Strings - Slicing

```
s[start:end:step]
```

- “abcdefg”[1:4:2] → “bd”
- “abcdefg”[::2] → “aceg”
- “abcdefg”[4:1:-1] → “edc”
- “abcdefg”[::-1] → “gfedcba”

Strings

```
def initials(first, last):
    return "first[0].last[0]"

first_name = input("What's your first name?")
last_name = input("What's your last name?")

print(initials(first_name,last_name))
```

→ Why is this not working properly… ?

Strings - Methods

Method	Description
`s.lower()`	Returns a lowercase copy of `s`
`s.upper()`	Returns an uppercase copy of `s`
`s.find(subs)`	Returns the index of `subs` in `s`
`s.startswith(prefix)`	Checks if `s` starts with `prefix`
`s.endswith(suffix)`	Checks if `s` ends on `suffix`
`s.strip()`	Returns a copy where whitespace at both ends have been removed
`s.lstrip()`	Returns a copy where whitespace at the start have been removed
`s.rstrip()`	Returns a copy where whitespace at the end have been removed
`s.ljust(width, fill)`	Returns a left justified copy of size `width` padded with `fill`
`s.rjust(width, fill)`	Returns a right justified copy of size `width` padded with `fill`

→ *-1 if not found*

Strings

```
def name(mailaddressUCLL):
    index_at = mailaddressUCLL.find("@")
    index_dot = mailaddressUCLL.find(".")
    first = mailaddressUCLL[0:index_dot]
    last = mailaddressUCLL[index_dot+1:index_at]
    return f"{first} {last}"

print(name("jef.janssens@ucll.be"))
```

→ Why is this not working properly… ?

Strings

```
def name(mail_UCLL):
    index_at = mail_UCLL.find("@")
    index_dot = mail_UCLL.find(".")
    first = mail_UCLL[0:index_dot]
    first = first[0].upper() + first[1:]
    last = mail_UCLL[index_dot+1:index_at]
    last = last[0].upper() + last[1:]
    return f"{first} {last}"
```

→ How do we capitalize first and last name?

Strings – DEMO

We consider an email address valid when it meets the following *simplified* requirements:

- It must contain an @, which cannot be at the first position
- There must be at least one letter right after the @
- The address must end with a . followed by exactly 2 characters

Write a function **valid_email(email)** that returns a **string** that tells us whether the given email address is valid or invalid as follows: “<<email>> is valid” or “<<email>> is invalid”.

Questions?

git pull

Strings – DEMO

We consider an email address valid when it meets the following *simplified* requirements:

- It must contain an @, which cannot be at the first position
- There must be at least one letter right after the @
- The address must end with a . followed by exactly 2 characters

Write a function **valid_email(email)** that returns a **string** that tells us whether the given email address is valid or invalid as follows: “<<email>> is valid” or “<<email>> is invalid”.