HSLU Lucerne University of Applied Sciences and Arts

Python for Data Science: SW02

Code structure
if-else statement
match-case statement

Information Technology
February 25, 2025

FH Zentralschweiz

Content

- Sequential data types (repetition)
- Type casting
- Input and output
- Boolean operation
  - And &; or |; not !
- Indentation and syntax
- If-else statement
  - Concept and components
  - Shorthand statement
  - Elif concatenation
- Match case

HSLU Lucerne University of Applied Sciences and Arts

Sequential data types (repetition)

Information Technology

February 25, 2025

Sequential data types (repetition)

In Python, sequential data types may comprise **mixed data types** and can have **multiple dimensions**.

Type	Example for assignment	Properties
List	`li = [1,'hello',2,[4,6,'a'],(3,9)]`	ordered; changeable
Tuple	`tp = (77,(55,1),9,37) or tp = 77,(55,1),9,37`	ordered; unchangeable
Dictionary	`di = {'a':1, 'b':2, 'c':{'aa':11, 'bb':22}}`	key-value pair; ordered; changable; key must be string; no duplication of elements
Set	`se = {1,28,6,(18,88),11}`	no duplication of elements; unordered; unindexed; contains only hashable elements

HSLU Lucerne University of Applied Sciences and Arts

Type casting

Information Technology

February 25, 2025

Type casting

Depending on operation, same data have to appear in **different** types.
E.g. Concatenating strings require string representation of all concatenating elements: ~~“hello “ + 1 + “st world”~~

-> data of a particular type are **transformed** into a different data type.

int:
```
int(‘3’), int(0x13), int(3.141592), ~~int(3+3j), int(‘hello’)~~
# any numerical value in string or number format, excl. complex numbers
```

float:
```
float(‘3’), float(0x13), float(3), ~~float(3+3j), float(‘hello’)~~
# any numerical value in string or number format, excl. complex numbers
```

string:
```
str(3), str(0x13), str([1,2,3]), str({‘a’:5, ‘b’:3}), str(...)
# any object supporting string representation
```

bool:
```
bool(‘3’), bool(0x13), bool(-3.1415)
# everything except 0, “”, None, empty or False evaluates to True
```

complex:
```
complex(‘3’), complex(‘3+3j’), complex(3.1415)
# any numerical value in string or number format
```

HSLU Lucerne University of Applied Sciences and Arts

Input and output

Information Technology

February 25, 2025

FH Zentralschweiz

Input and output

Interaction with an application requires input and output.

Input options:

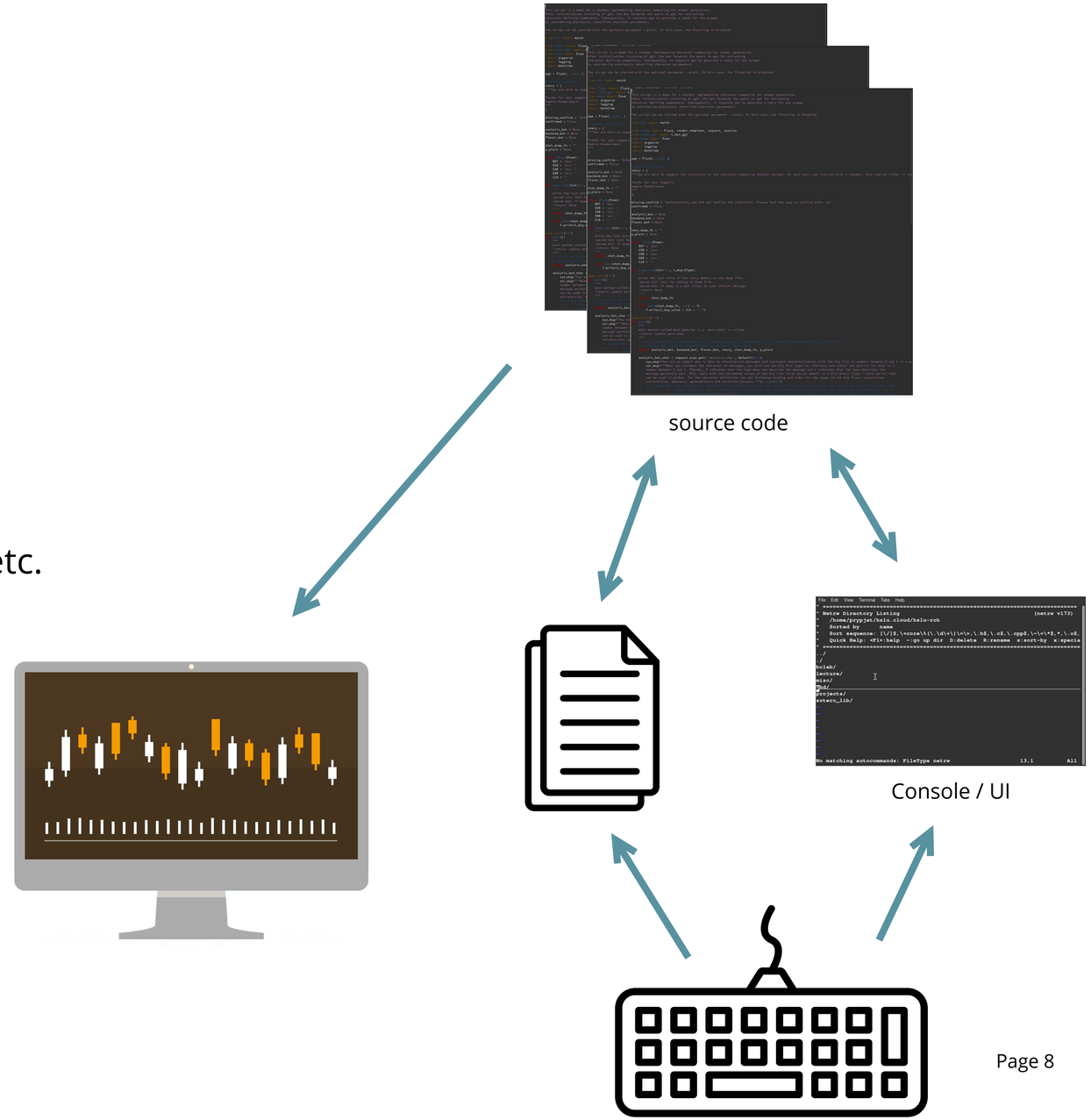
Camera, **File**, **Keyboard**, Microphone, Mouse, USB, etc.

Output options:

File, Speaker, **Terminal**, USB, User Interface (text, figures), etc.

Documentation:

https://docs.python.org/3/tutorial/inputoutput.html



Input and output

Reading from standard input using the function `input(prompt)`:
Doc: https://docs.python.org/3/library/functions.html#input

- The input function prints the passed string to the standard output (i.e. terminal)
  -> without trailing newline (“`\n`”).
- Reads from standard input (i.e. terminal) and converts it to string.
  -> no trailing newline (“`\n`”) at the end.
- `eval()` allowes to evaluate Python expressions. This is helpful when receiving expressions as input: `eval(“3+5”)`

```
>>> s = input('--> ')
--> Monty Python's Flying Circus
>>> s
"Monty Python's Flying Circus"
>>> s = input('integer? ')
integer? 44
>>> s
'44'
>>>
```

Input and output

Sending information to standard output using the function:

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

Doc: https://docs.python.org/3/library/functions.html#print

- The function sends all arguments in objects to the standard output (i.e. terminal).
- Parameters are:

`*objects`:	one or more **string** elements
`sep`:	string written in between of elements
`end`:	trailing string
`file`:	file object providing `write(string)` method
`flush`:	boolean if stream will be forcibly flushed

HSLU Lucerne University of Applied Sciences and Arts

Boolean operation

Information Technology

February 25, 2025

FH Zentralschweiz

Boolean operation

A boolean expression can be of two states only:

`0` or `1` **`left` or `right`** **`true` or `false`**

Doc: https://docs.python.org/3/library/stdtypes.html#boolean-operations-and-or-not

Python treats all that is NOT **empty**, **0, False** or **None** as **True**.

```
>>> bool(None)
False
>>> bool("")==bool(0)==bool(None)==False==(not True)
True
>>>
```

Boolean operation: or, and, not

or

- `x or y`
- Priority: 1
- if x is true, then x, else y

	True	False
True	True	True
False	True	False

and

- `x and y`
- Priority: 2
- if x is false, then x, else y

	True	False
True	True	False
False	False	False

not

- `not x`
- Priority 3
- if x is false, then True, else False
- `x == not y` -> syntax error
- `not x == y` equal as
  `not ( x == y )`

True -> False

False -> True

or: short-circuit operator; second argument is **only** evaluated if first one is **false**
and: short-circuit operator; second argument is **only** evaluated if first one is **true**

Boolean operation: operators

Bitwise operators

Operation	Result
`x	y`	bitwise or of x and y
`x ^ y`	bitwise exclusive or of x and y
`x & y`	bitwise and of x and y
`x << n`	x shifted left by n bits
`x >> n`	x shifted right by n bits
`~x`	the bits of x inverted

Comparison operators

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Comparisons can be chained arbitrarily: `x < y <= z` is equivalent to `x < y and y <= z`
`z` is not evaluated at all when `x < y` is found to be **false**.

HSLU Lucerne University of Applied Sciences and Arts

Indentation and syntax

Information Technology

February 25, 2025

FH Zentralschweiz

Indentation and syntax

In Python, the code structure is mainly given by **indentations** and **priorities**.

Boolean operations:

- Structure is given by operation priorities.
- How does the code have to be changed to become **True**?

```
>>> True and False or False and True or False
False
>>>
```

- Priorities can be changed by grouping in brackets:

```
True and (False or False) and (True or False)
```

- Boolean operations allow for **combining** conditions.
  e.g. matching even number in range: 12 < x < 20

```
12 < x < 20 and not x % 2
```

Indentation and syntax

Block operations start with a column (:) and are defined by indentations:
-> indentations can have arbitrary **number of spaces** but must be **constant** for all instructions of the same block content
Doc: https://docs.python.org/3/reference/lexical_analysis.html#indentation

Single block

```
instruction
instruction
block header:
    block instruction
    block instruction
    block instruction
instruction
instruction
```

Nested block(s)

```
instruction
instruction
block 1 header:
    block 1 instruction
    block 2 header:
        block 2 instruction
        block 2 instruction
    block 1 instruction
    block 1 instruction
instruction
instruction
```

HSLU Lucerne University of Applied Sciences and Arts

If-else condition

Information Technology

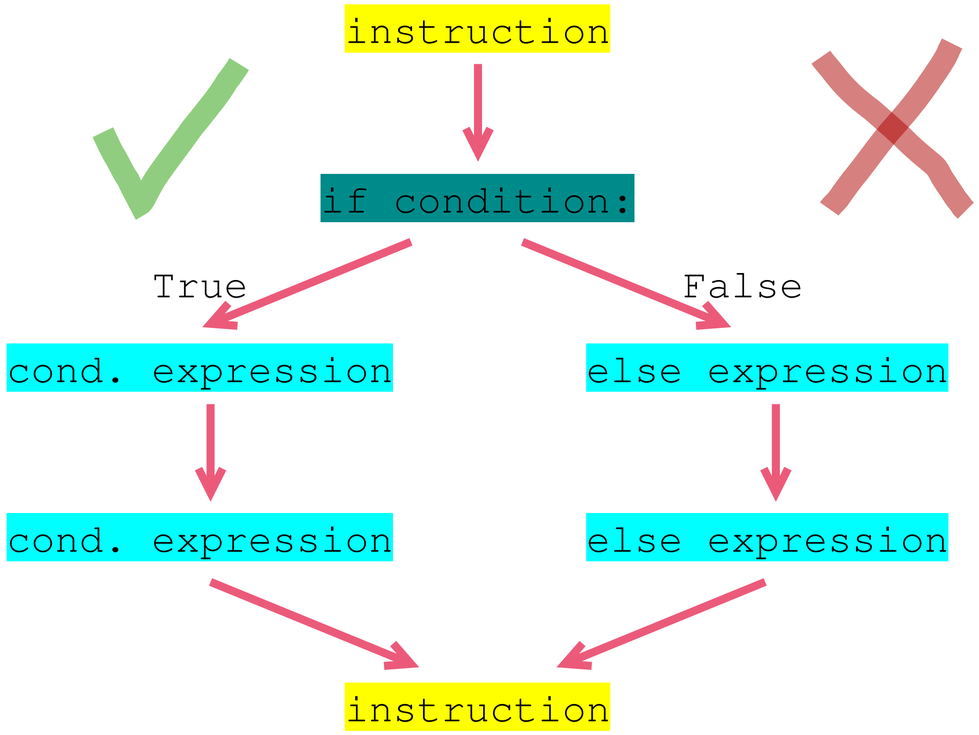
February 25, 2025

FH Zentralschweiz

If-else statement: Concept and components

The if-else statement is a switch for proving and catching a condition being required for a conditional expression.

- It allows for executing expressions (a block) under certain conditions (if condition is True).
- The else block is **not** mandatory.



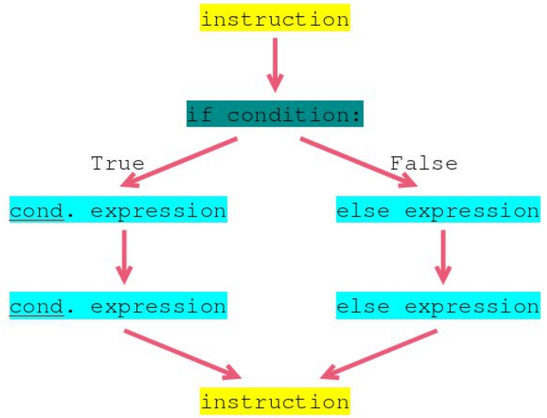

```
instruction
if condition:
    cond. expression
    cond. expression
else:
    else expression
    else expression
instruction
```

If-else statement: Concept and components

Following an example switching between conditional expressions:

```
x = 7                   # x = 7
if x > 10:              # 7 > 10 -> condition = False
    print(x)            # not executed
else:                   # switch to else
    x = x + 5           # x = 12
print(x)                # print 12 to console

if x > 10:              # 12 > 10 -> condition = True
    print(x)            # print 12 to console
else:                   # no switch to else
    x = x + 5           # not executed
print(x)                # print 12 to console
```



If-else statement: Elif concatenation

In the regular if-statement expression, elif allows for concatenating conditions.

- **elif** brings the **nested** if-statement to the same level of the previous if-statement.
- elif can always be used as replacement for the else statement to bring in another condition.

```
x = 30
if not x % 2:                    # True if x not even number
    x = x + 1                    # increment x
elif x < 10:                     # True if x < 10
    x = x                        # parity
else:                            # x is even number and x >= 10
    x = x / 10
```

If-else statement: Shorthand statement

Frequently, conditional statements only affect a single expression. In this case, shorthand if-statements allow for less code and better readability.

```
x = 30
x = x + 1 if x % 2 else x                    # increase to next even number if x == odd
```

Shorthand if-statements can also be concatenated:

```
x = x + 1 if x % 2 else x if x < 10 else x / 10 # increase to next even number if x == odd else
                                                # divide by 10 if x >= 10
```

Shorthand if-statement also applicable with function calls:

```
print('hello world') if x % 2 else print('end')
```

HSLU Lucerne University of Applied Sciences and Arts

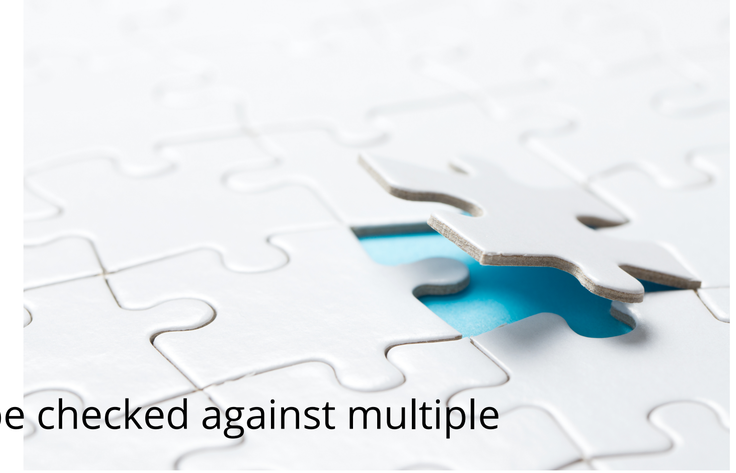
Match case

Information Technology

February 25, 2025

FH Zentralschweiz

Match case



As an alternative to multiple combined conditions with elif-statements, an expression can be checked against multiple **particular cases** to which it matches.

- The **default case** is given with an underscore: _
- A match breaks the statement. All other cases are not proved.
- Doc: https://docs.python.org/3/tutorial/controlflow.html#match-statements

Elif-statement

```
x = 2
if x == 0:
    print(‘number is 0’)
elif x == 1:
    print(‘number is 1’)
elif x == 2:
    print(‘number is 2’)
else:
    print(‘number unknown’)
```

Match-case-statement

```
x = 2
match x:
    case 0:
        print(‘number is 0’)
    case 1:
        print(‘number is 1’)
    case 2:
        print(‘number is 2’)
    case _:
        print(‘number unknown’)
```

Match case: guard clause

Match case statements prove against particular expressions and not conditions. Despite, guard clause allows to prove against conditions. Usually, this case is solved with elif-statements.

Elif-statement

```
x = 177
e = 0
if 100 > x >= 10:
    e = 1
elif 1000 > x >= 100:
    e = 2
elif x >= 1000:
    e = 3
else:
    print(‘irregular number’)
print(x/(10**e), “E+”, e)
```

Match-case-statement

```
x = 177
e = 0
match x:
    case x if 100 > x >= 10:
        e = 1
    case x if 1000 > x >= 100:
        e = 2
    case x if x >= 1000:
        e = 3
    case _:
        print(‘irregular number’)
print(x/(10**e), “E+”, e)
```

HSLU Lucerne University of Applied Sciences and Arts

School of Computer Science and Information Technology
Research
Ramón Christen
Research Associate Doctoral Student

Phone direct +41 41 757 68 96
ramon.christen@hslu.ch

HSLU T&A, Competence Center Thermal Energy Storage
Research
Andreas Melillo
Lecturer

Phone direct +41 41 349 35 91
andreas.melillo@hslu.ch