Industrial Software Development

Tutoring sessions

Tutor: Maura Pintor (maura.pintor@unica.it)

Organization

Thursday h 15:00 - 18:00 Room AB and Teams channel

Topics:

1. Setup and Python basics
2. Python advanced
3. Design patterns (required for exam)
4. Projects in teams

Organization

All lessons will be held in hybrid mode. The first three topics will be also recorded and available on Teams. The group activity will not be recorded (and in-presence mode is recommended!).
Ask questions! Use the Teams channel, send me a PM on Teams (associated to this email: **maura.pintor@unica.it**), or directly send an email to **maura.pintor@unica.it**

Today: Setup and Python basics

What to know about Python:

- it is an **interpreted language**
- indentation matters
- extremely flexible (not always a good thing)

Setting up the interpreter

1. Install **Miniconda**
2. Create an interpreter `conda create --name isde`
3. Activate the interpreter `conda activate isde`
4. Use the interpreter
5. (optional) Deactivate the interpreter `conda deactivate`
   The last step is optional as it will be done automatically when the shell session terminates.

Setting up the interpreter

If you have an IDE, check out the dedicated instructions for setting the interpreter in your project. It will make life easier!

Python variables and types

```
# examples of types
number = 100  # integer
another_number = 10.6  # floating point
letter= "a"  # char

# gets the type of a variable
type(number)

str(number)  # type conversion
```

Exercise

Print the results of the following expressions:

```
a = 15
b = -20
c = a-b
c = not a
c = (a==b) or ((a+b)==-5)
```

Exercise

Try to predict what will be the result of these expressions

```
print ((4>6) and (3>8))
print ((4>6) or (3>8))
print (1 == 1 and 2 == 1)
print (False and 0 != 0)
print (not (True and False))
```

Aggregations of variables

```
l = [0, 1, 2]  # list
s = "some text"  # string

print(type(l))
print(type(s))

l = [[0, 1], [2, 3]]  # nesting
```

Indexing lists and strings

```
l = [0, 1, 2]
```

```
# take first element (or element at position X)
fist_element = l[0]  # numbering starts from zero!
first_letter = s[0]

# take last element (or element at position -X)
last_letter = s[-1]
```

Indexing lists and strings

```
# take portion of elements (from A to B -> B not included)
group_of_letters = s[1:5]  # elements from 1 to 4

# if one end is not specified, uses a default
first_letters = s[:2]  # default: start from 0
last_letters = s[2:]  # default: end with -1

# take one element each N
reverse_string = s[::2]  # one each 2

# take elements in reverse
reverse_string = s[::-1]
```

Indexing lists and strings

```
l = [[1, 2], [3, 4, 5]]

# access nested lists
sublist = l[0]
element_of_sublist = l[0][1]
```

Edit elements

```
l = ['a', 1, 2]
l[0] = 0  # changes first element to 0
```

REMEMBER: elements of strings cannot be edited!

```
s = "text"
s[0] = "n"  # DOES NOT WORK
s = "next"  # re-defining the string works
```

Operations with strings and lists

```
l = [0, 1, 2]
l = l + [3, 4, 5]  # concatenates the lists
s = "abc" + "def"  # works also with strings

s = '!' * 30  # creates a string with 30 '!'

number_of_chars = len(s)  # length of the string
```

Operations with strings only

```
s = "abCdef"
lowercase = s.lower()
uppercase = s.upper()

# checks if sequence "de" is contained in the string
de_in_s = "de" in s  # case sensitive!
```

Fancy strings

Since Python 3.6, strings can access directly python variables.

```
number = 222
s = f"The lucky number is {number}"
print(s)
```

This converts the variable automatically into a string type and concatenates it inside of the string.

Tuples

Tuples can be seen as unmutable lists.

```
t = (0, 1, 2,)

t[0] = 1  # does not work
```

Dictionaries

```
d = {'age': 22, 'name': 'joe'}
age = d['age']  # takes element by key
keys = d.keys()  # all keys
values = d.values()  # all values
d['surname'] = 'doe'  # adds one key
d['name'] = 'john'  # edits element
```

Remember: each key is unique! Remember: keys are not sorted!

Sets

```
# stores unique values, removes duplicates
s = {1, 2, 3, 1, 2, 3}
```

As dictionaries, the sets don't preserve the order of the elements.

Control structures

How to change the control flow of a program?

- `if`
- `if-else`
- `if-elif-else`
- `while`
- `for`

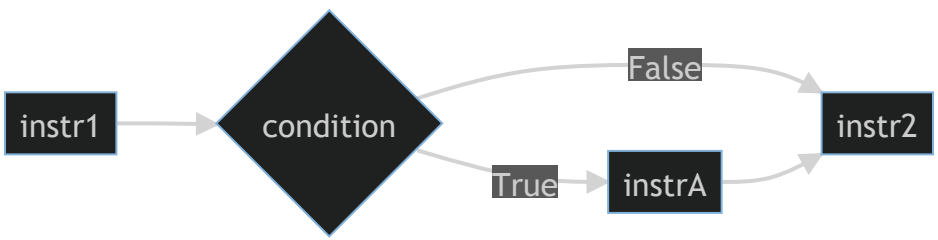
Remember: Python uses intentation to define control flow structures

Normal control flow

```
a = 2  # instr1
b = a + 3  # instr2
c = b/2  # instr3
```

If-only control flow

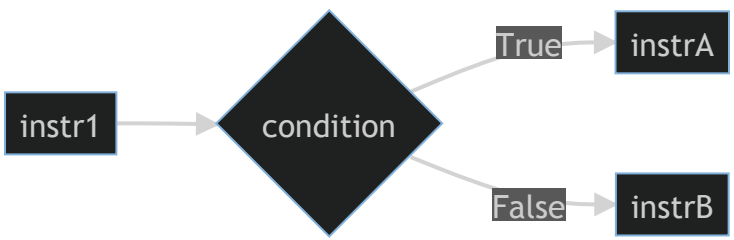
```
a = 2  # instr1
if a >= 1:  # condition
    print("greater than or equal to 1")  # instrA
print(a)  # instr2
```



ATTENTION: anything different than `0` or `False` is considered `True`!

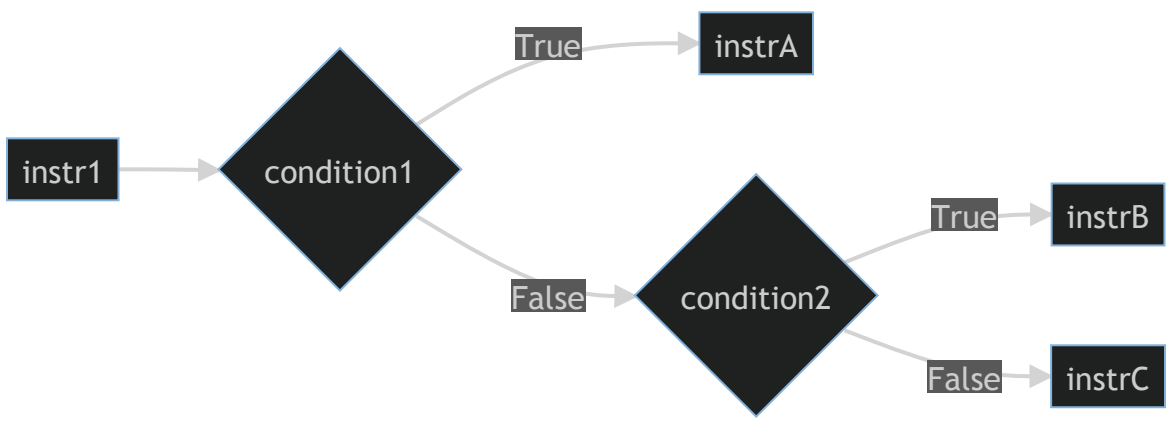
If-else control flow

```
a = 2  # instr1
if a >= 1:  # condition
    print("greater than or equal to 1")  # instrA
else:
    print("smaller than 1")  # instrB
```



If-elif-else control flow

```
a = 2  # instr1
if a >= 1:  # condition1
    print("greater than 1")  # instrA
elif a == 1:
    print("equal to 1")  # instrB
else:
    print("smaller than 1")  # instrC
```



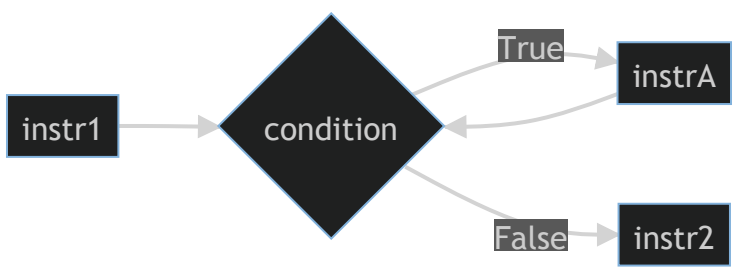
Loops

What if we have to repeat the same instruction multiple times? We can use loops. They repeat the instruction until the stop condition is met.

While loops

Used to repeat actions until condition is met.
`CTRL + c` if you get stuck in a while loop :D

```
a = 0  # instr1
while a < 10:  # condition
    a += 1  # instrA
print(a)  #inst2
```



For loops

Used for index-based structures or variables or to repeat actions (exactly) N times.

```
#get only items
for item in [0, 1, 2]:
    print(item)
```

```
# get index and item
for index, item in enumerate([0, 1, 2]):
    print(index,item)
```

```
# do action exactly N times
for i in range(10):
    print("string")
```

For loops

Can be used to iterate through dictionaries.

```
d = {'A': [0, 1, 2], 'B': [3, 4, 5]}

for key, value in d.items():
    print(key, value)
```

Exercise

Print the sum of a list of integers.
Implement it with a `for` loop and a `while` loop.

Functions

Useful for dividing complex problems into more simple and solvable sub-problems. They are blocks of code that solve a sub-problem, returning directly the result.

```
students = {
    'A': [20, 23, 22, 19],
    'B': [18, 22, 19, 28],
    'C': [30, 18, 22, 30]
    }
```

- Main problem: Take away one point for each score of each student.
- Sub-problem: Subtract one to all elements of a list (solved for each student).
- Sub-sub-problem: Subtract one to one element.

Functions

- Input parameter(s): information required to find the result (`x`)
- Output(s): result of the function (`return x - 1`)

```
def subtract_one(x):
    """Subtracts one point from the score x"""
    return x - 1

def subtract_one_list(x):
    """Returns a new list with the scores lowered by 1"""
    new_list = []  # empty list
    for item in x:
        lower_score = subtract_one(item)
        new_list.append(lower_score)  # appends element
    return new_list

def subtract_one_to_all(x):
    ...  # have fun :D
```

Functions

Functions can take no input or return no output.

```
def no_input():
    return 10

def no_return(x):
    print(x)

def no_input_and_return():
    print("I will return nothing.")

result = no_input_and_return()  # returns None
```

Functions

Functions can take multiple parameters as input and return multiple results.

```
# two input parameters
def concatenate_strings(s1, s2):
    return s1 + s2

# return two results
def sum_and_subtract_10(x):
    return 10+10, 10-10
```

Scope

The variables inside the function are only accessible within the function. This is referred as **scope**.

```
# two input parameters
def concatenate_strings(s1, s2):
    print(s1)  # works

a = "aa"
b = "bb"
concatenate_stings(a, b)
print(s1)  # gives error
```

Exercise

Write four functions that perform the mathematical operations sum, product, division, subtraction. They have to take as input two numbers and return the result (don't call them the same as built-in functions!).

Built-in functions

Python has many built-in functions.

```
x = [0, -1, 2, -10]
m = min(x)  # finds the minimum
l = len(x)  # finds length
```

RECOMMENDED: DO NOT CALL VARIABLES OR FUNCTIONS WITH BUILT-IN FUNCTION NAMES!!!

Libraries

```
import math
x = 0
s = math.sin(x)

import math.sin
s = math.sin(x)  # imports only the module 'sin'

from math import sin
s = sin(x)  # can be used without specifying the library
name

import math as m
s = m.sin(x)  # uses an alias for the library
```

Exercise

Write a snippet that computes the area of a circle, given the radius:

```
import math.pi

...
```

Install libraries

External libraries can be installed through the python packet managers. The most used ones are conda and pip.

```
conda install pip
pip install numpy
```