FRUIZIONE E UTILIZZO DEI MATERIALI DIDATTICI

- **E’ vietata** la **copia**, la **rielaborazione**, la **riproduzione** dei contenuti e immagini presenti nelle lezioni in qualsiasi forma

- **E’ inoltre vietata** la **diffusione**, la **redistribuzione** e la **pubblicazione** dei contenuti e immagini, incluse le registrazioni delle videolezioni con qualsiasi modalità e mezzo non autorizzati espressamente
dall'autore o da Unica

Dispatch, lookup tables and other technics

Polymorphism is the provision of a single interface to entities of different types[1].
Polymorphism allows a programming language to decide at runtime which method to use based on the parameters sent to the method.

```
#Example: the implementation of `f()`depends on the object class.
obj = A() # B()
obj.f()
```

Dynamic dispatch is the process of selecting which implementation of a polymorphic operation (method or function) to call at run time. The choice of which version of a method to call may be based either on a single object/parameter (single dispatch), or on a combination of objects (multiple dispatch).
- **Single dispatch** is a type of polymorphism where only one parameter is used to determine the call. Usually, this parameter is the object itself (**self**, or **this**).
- **Multiple dispatch** is a type of polymorphism where multiple parameters are used in determining which method to call.

---

[1] https://www.stroustrup.com/glossary.html#Gpolymorphism

Example:

```
dividend.divide(divisor) #a/b
```

Single dispatch: the implementation will be chosen based only on dividend's type (rational, floating point, matrix,..), disregarding the type or value of divisor.

Multiple dispatch: the implementation will be chosen based on the combination of operands. The types of the `dividend` and `divisor` together determine which `divide` operation will be performed.

EXERCISE

Write the classic `rock-paper-scissor` [2] game. Rock, paper and scissor could be instance of the classes `Rock` , `Paper` , `Scissor` , respectively (but this is not the only possibility)

Is it possible to avoid an `if` sequence?

Think about how to extend the game to other weapons, as in "rock paper scissors Spock lizard"[3]

---

[2] https://en.wikipedia.org/wiki/Rock_paper_scissors

[3] https://en.wikipedia.org/wiki/Rock_paper_scissors#Additional_weapons

Note (code in `dispatch/rock_paper_scissor_*`)

- `rock_paper_scissor_1` is the traditional approach using a conditional structure.
- `rock_paper_scissor_2` and `rock_paper_scissor_3` uses a lookup table. `rock_paper_scissor_2` shows that classes can be used as keys in a lookup table (a dictionary). `rock_paper_scissor_3` uses a straightforward approach, using a single class and a variable indicating the type of 'weapon'.
- `rock_paper_scissor_4` presents a simple example of 'double dispatch'. This is not the best solution in this case - it is a complex solution to a simple problem - but it is useful to analyze the mechanism in simple cases.

Exercise - Double dispatch

R is the class of real number, and **C** is the class of the complex number[4].

```
# code in
# `dispatch/real_complex_01`
class MathEntity(ABC):
    ...
class R(MathEntity):
    def __init__(self, a):
        self.a = a

class C(MathEntity):
    """represents a + i b """
    def __init__(self, a, b):
        self.a = a
        self.b = b
```

Write methods to sum and multiply instances of `R` and `C` using
- 'double dispatch' approach
- 'lookup-table' approach
- 'if - elif - else' approach with explicit type checking.

Discuss the differences.
Extend the problem adding quaternions[5] and array.

---

[4] This is a toy problem - real and complex are already implemented in python.

[5] Quaternions are a number system that extends the complex numbers. They are in the form $a + b\,\mathbf{i} + c\,\mathbf{j} + d\,\mathbf{k}$. See https://en.wikipedia.org/wiki/Quaternion

We can use this approach when the behavior depends on the pair of objects involved in the relationship, and we plan to extend the number of classes involved.

Example: the software manages different displays and different geometric shapes. The way to draw a geometric figure depends both on the figure itself and on display.

```
s1 = Polygon()
s2 = Ellipse()
d1 = Display1(h=640, l=640, bit=16)

s1.display_on(d1)
```

A solution could be the following.
- Each display knows how to draw different shapes, so it implements `display_polygon(shape)`, `display_ellipse(shape)`.
- Each shape has a method `display_on(display)` which it uses to send itself to the display.

You can choose to write che concrete drawing algoriothm in the shape classes, in the display classes, or in a third class.

Example: a game in which the characters fight each other or team up with each other, and the result depends on their class (elves, sorcerers, etc.) and other parameters.

```
Wizard + wizard:
print -> (name1, name2) - now our strength is: (the sum of their energy level)

elf + elf:
print -> (name1, name2) - now our strength is: (the double of the sum of their energy level)

elf + wizard:
print -> we are not compatible

knight + wizard:
print -> (name1, name2) - now our strength is: (wizard energy + 2 * knight energy)
(And so on)
```

Implement only `wizard` and `elf`. After this, add `knight`.

- Use the if-elif-else approach. If you try to add other characters, you need to change the conditional logic.
- Use the double-dispatch method. You can add new classes and add new behavior following the OPEN-CLOSED principle.

References

- Muschevici, Radu, et al. "Multiple dispatch in practice." Acm sigplan notices 43.10 (2008): 563-582. https://dl.acm.org/doi/pdf/10.1145/1449764.1449808
- Bezanson, Jeff, et al. "Array operators using multiple dispatch: A design methodology for array implementations in dynamic languages." Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming. 2014. https://dl.acm.org/doi/pdf/10.1145/2627373.2627383