

---

---

# Interpreter Pattern

Grupo 06

Gustavo

João Henrique

---

---

# Definição

Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

- Gang of Four

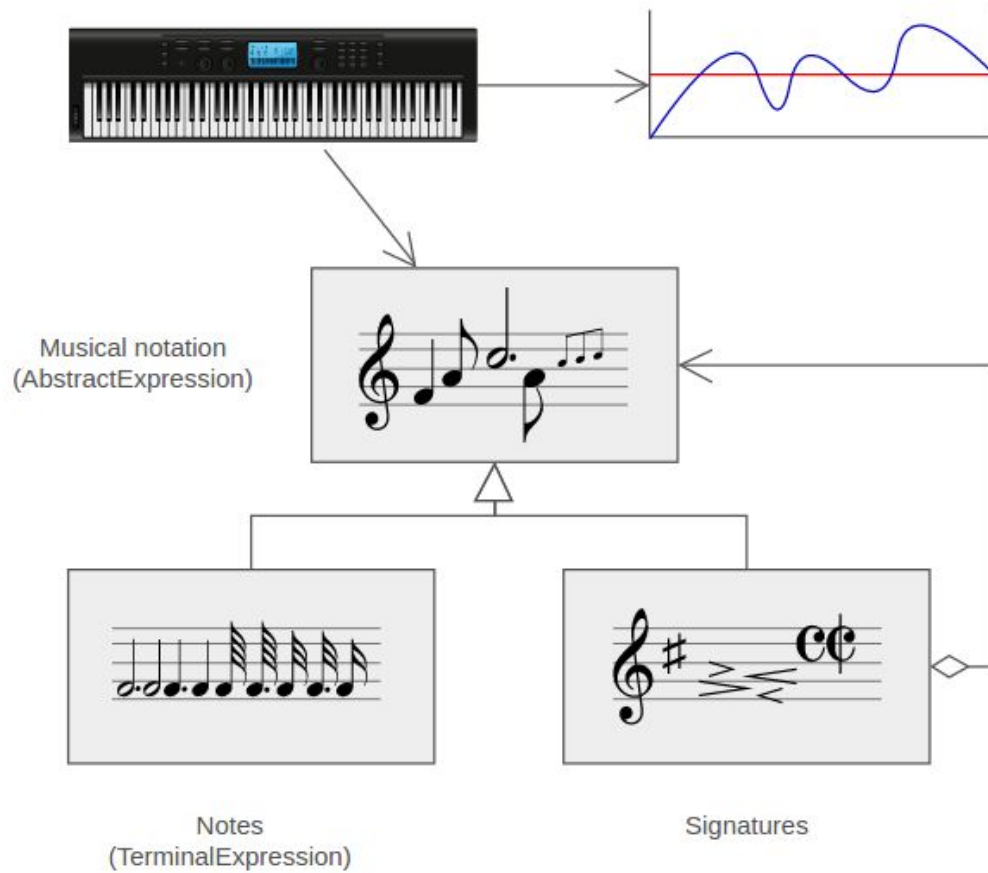
Dada uma linguagem, definir uma representação para sua gramática juntamente com um interpretador que usa a representação para interpretar sentenças dessa linguagem.

# Responsabilidades do Interpreter Pattern

- Gerir algoritmos,
- Relações
- Responsabilidades entre os objetos

# No mundo real

O exemplo mais didático é de um tradutor, permitindo que a partir de uma entrada as pessoas possam entender uma língua estrangeira. Outro bom exemplo são os músicos, que usam a notação musical, atuando como intérpretes, na reprodução da música .





# Aplicação em código

## Problema

- Converter uma String (número romano) -> um inteiro que represente seu valor decimal

## Escolha de implementação

-  Percorrer a String e procurar cada um dos possíveis casos não é a melhor solução
-  Vamos tentar formular o problema como uma gramática

# O que é gramática

Conjunto de leis de formação que definem de maneira rigorosa, o modo de geração de textos corretos de uma linguagem.

# Definindo a gramática

Um número romano é composto por caracteres que representam números de quatro, três, dois ou um dígito:

- número romano ::= {quatro dígitos} {três dígitos} {dois dígitos} {um dígito}

-



Números de quatro, três, dois e um dígito são formados por caracteres que representam nove, cinco, quatro e um. Com estes caracteres é possível representar qualquer um dos número em romanos:

- 1: I
- 2: II
- 3: III
- 4: IV
- 5: V
- 6: VI
- 7: VII
- 8: VIII
- 9: IX

# Definindo a Gramática

Números de quatro, três, dois e um dígito são formados por caracteres que representam nove, cinco, quatro e um. Com estes caracteres é possível representar qualquer um dos número em romanos:

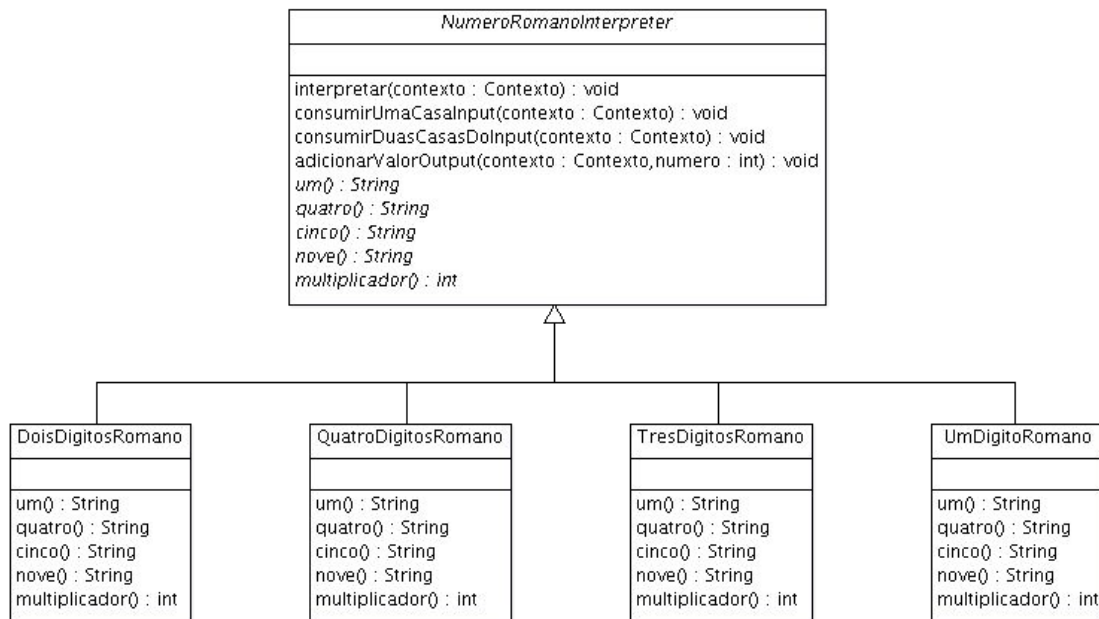
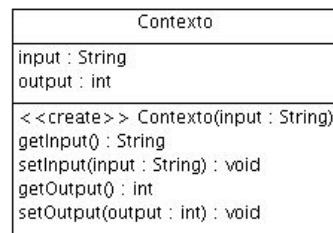
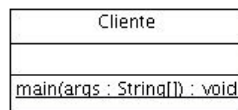
quatro dígitos ::= um

três dígitos ::= nove | cinco {um} {um} {um} | quatro | um

dois dígitos ::= nove | cinco {um} {um} {um} | quatro | um

um dígito ::= nove | cinco {um} {um} {um} | quatro | um

# Implementação



```
1 class Context():
2     _input = ""
3     _output = 0
4
5     def __init__(self, input_context):
6         self._input = input_context
7
8     def get_input(self):
9         return self._input
10
11     def set_input(self, _input):
12         self._input = _input
13
14     def get_output(self):
15         return self._output
16
17     def set_output(self, _output):
18         self._output = _output
```

```
3 class OneDigit(Interpreter):
4
5     def one(self):
6         return "I"
7
8     def four(self):
9         return "IV"
10
11     def five(self):
12         return "V"
13
14     def nine(self):
15         return "IX"
16
17     def weight(self):
18         return 1;
19
```

```

4 class Interpreter():
5     __metaclass__ = ABCMeta
6
7     def to_interpret(self, context):
8
9         if len(context.get_input()) == 0:
10             return
11
12         if context.get_input().startswith(self.nine()):
13             self.add_value_output(context, 9)
14             self.consume_digits(context, 2)
15         elif context.get_input().startswith(self.four()):
16             self.add_value_output(context, 4)
17             self.consume_digits(context, 2)
18         elif context.get_input().startswith(self.five()):
19             self.add_value_output(context, 5)
20             self.consume_digits(context, 1)
21
22         while context.get_input().startswith(self.one()):
23             self.add_value_output(context, 1)
24             self.consume_digits(context, 1)
25
26     def consume_digits(self, context, digits):
27         context.set_input(context.get_input()[digits:])
28
29     def add_value_output(self, context, number):
30         context.set_output(context.get_output() + number * self.weight())

```

```

32 @abstractmethod
33 def weight(self): pass
34
35 @abstractmethod
36 def one(self): pass
37
38 @abstractmethod
39 def four(self): pass
40
41 @abstractmethod
42 def five(self): pass
43
44 @abstractmethod
45 def nine(self): pass

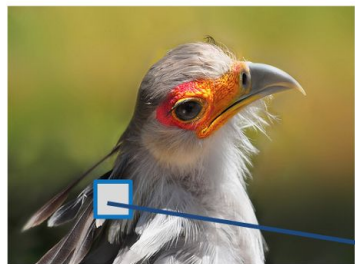
```

```
8  if __name__ == '__main__':
9
10     digits = [FourDigits(), ThreeDigits(), TwoDigits(), OneDigit()]
11
12     context = Context(str(input("")))
13
14     for d in digits:
15         d.to_interpret(context)
16
17     print (context.get_output())
```

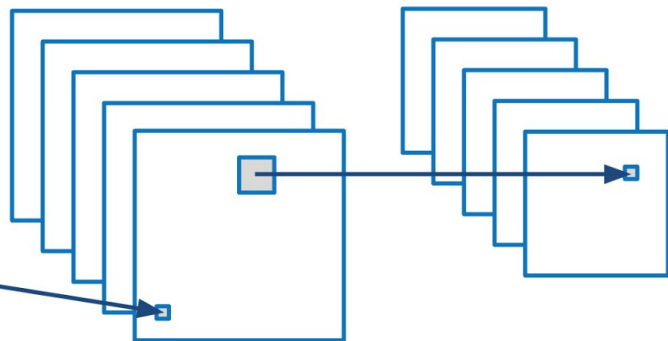
<https://github.com/gutioliveira/DesignPatterns/tree/master/Interpreter-Python/roman>

# Contexto do trabalho (processamento de imagens)

Construído com base em redes neurais que são modelos computacionais inspirados pelo sistema nervoso que são capazes de realizar o aprendizado de máquina bem como o reconhecimento de padrões. Redes neurais artificiais geralmente são apresentadas como sistemas de "neurônios interconectados que podem computar valores de entradas".

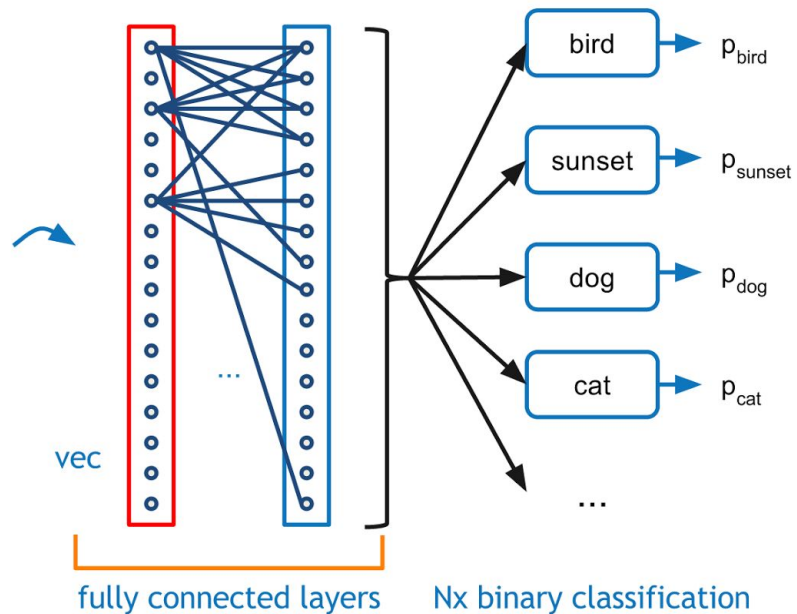


convolution +  
nonlinearity



max pooling

convolution + pooling layers



vec

fully connected layers

Nx binary classification



# Referências

- <https://brizeno.wordpress.com/category/padroes-de-projeto/interpreter/>
- [https://en.wikipedia.org/wiki/Deep\\_learning](https://en.wikipedia.org/wiki/Deep_learning)
- <http://deeplearning.net/>
- <https://dzone.com/articles/design-patterns-uncovered-14>
- <http://c2.com/cgi/wiki?InterpreterPattern>
- [https://sourcemaking.com/design\\_patterns/interpreter](https://sourcemaking.com/design_patterns/interpreter)
- <https://github.com/BVLC/caffe>
  - [DIY Deep Learning for Vision with Caffe \(Slide\)](#)