Luciano Ramalho
luciano@ramalho.org

oficinas Turing

dezembro/2012

Objetos Pythonicos

Orientação a objetos e padrões de projeto em Python

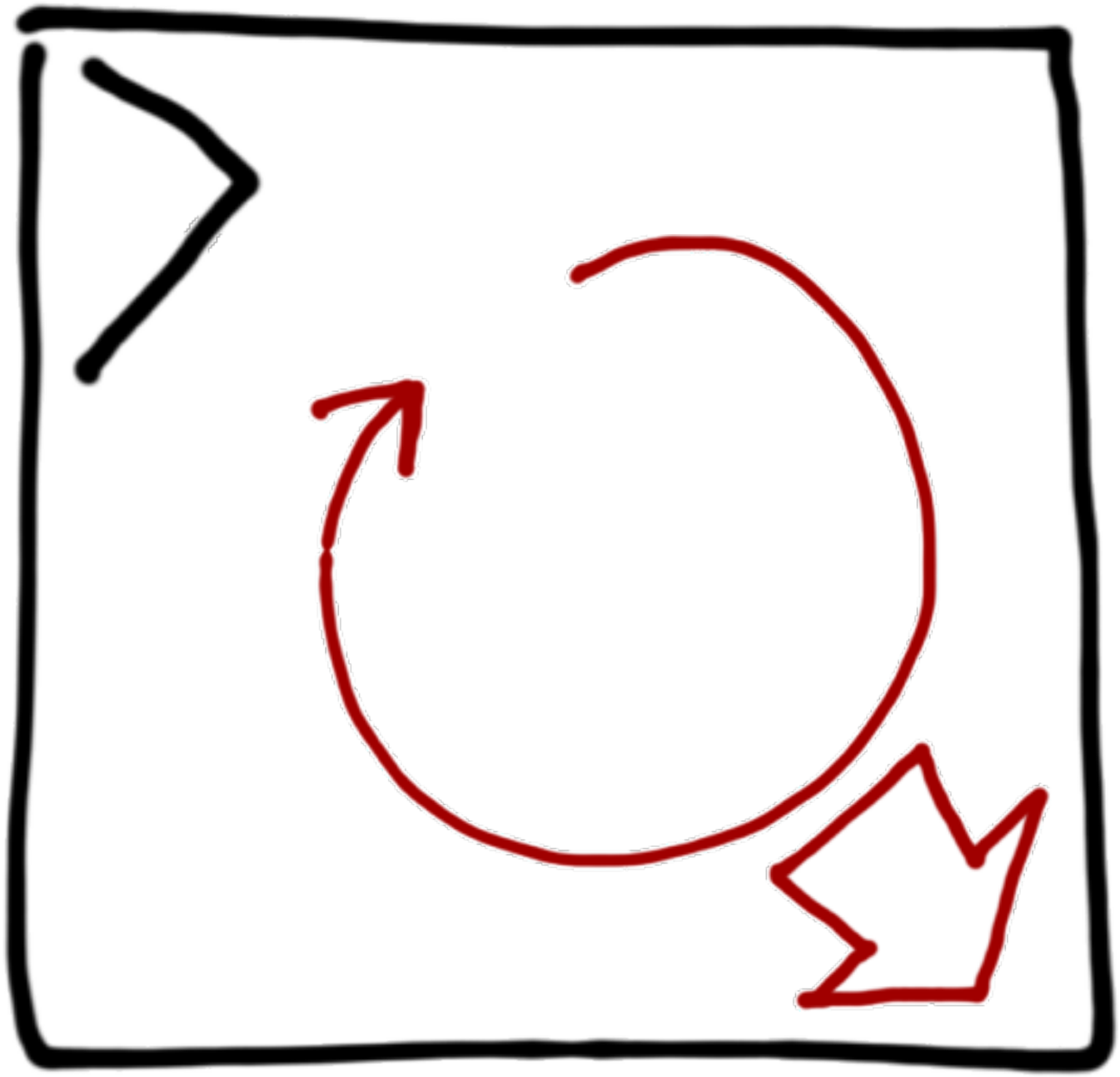
Exemplo prático com funções geradoras

- Funções geradoras para desacoplar laços de leitura e escrita em uma ferramenta para conversão de bases de dados semi-estruturadas

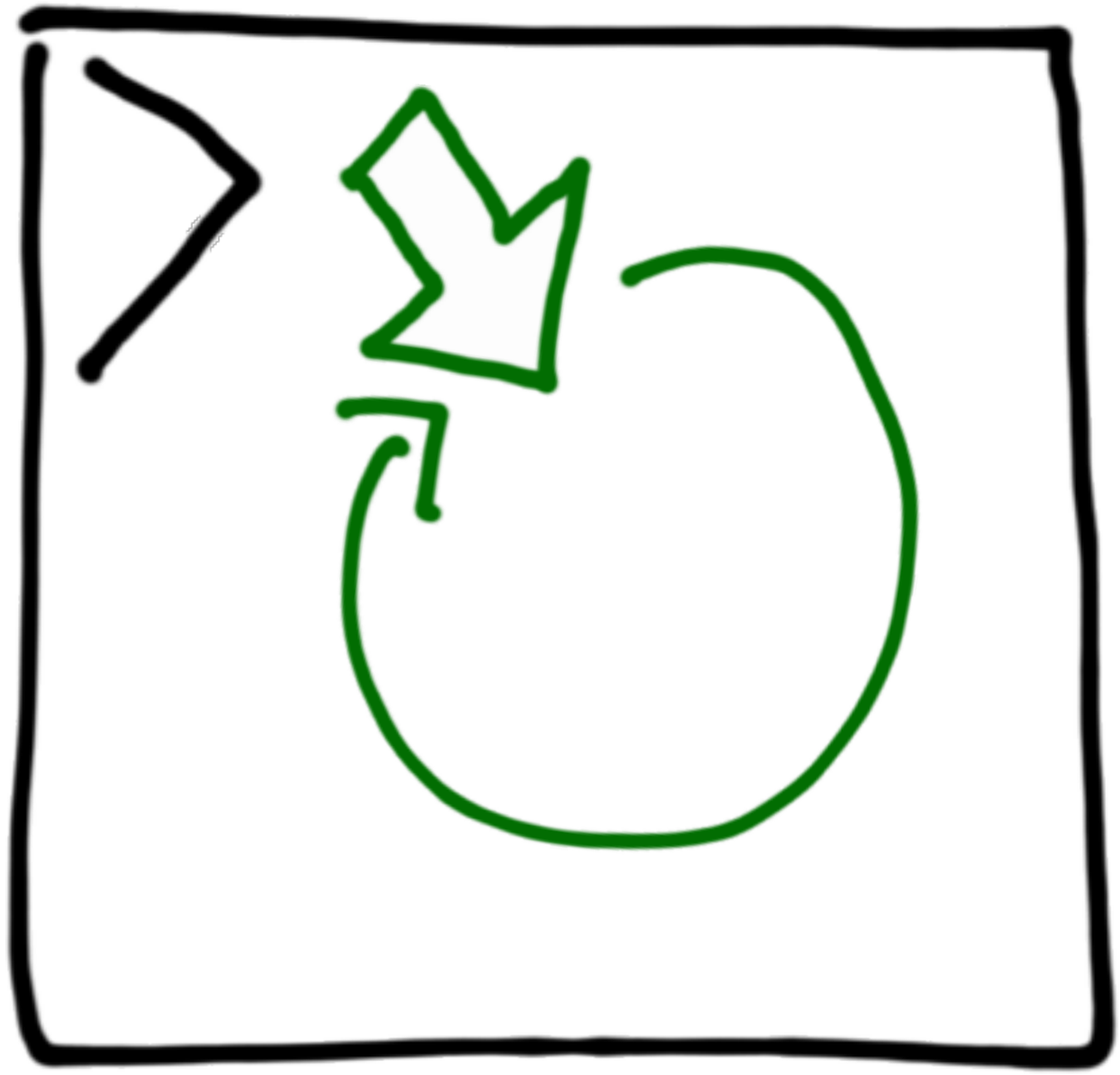
`https://github.com/ramalho/isis2json`

Exemplo:
funções geradoras

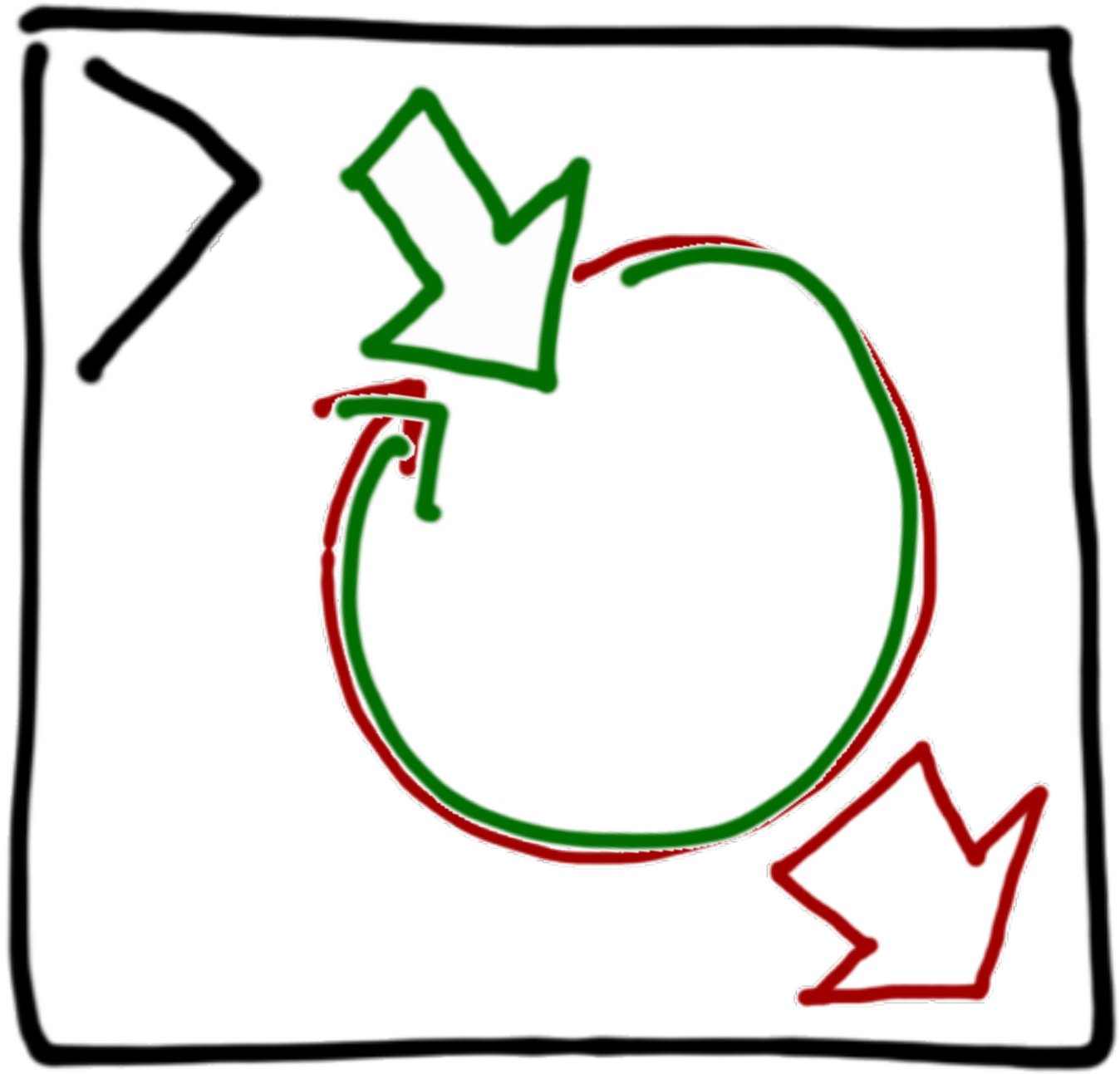
Laço principal escreve arquivo JSON



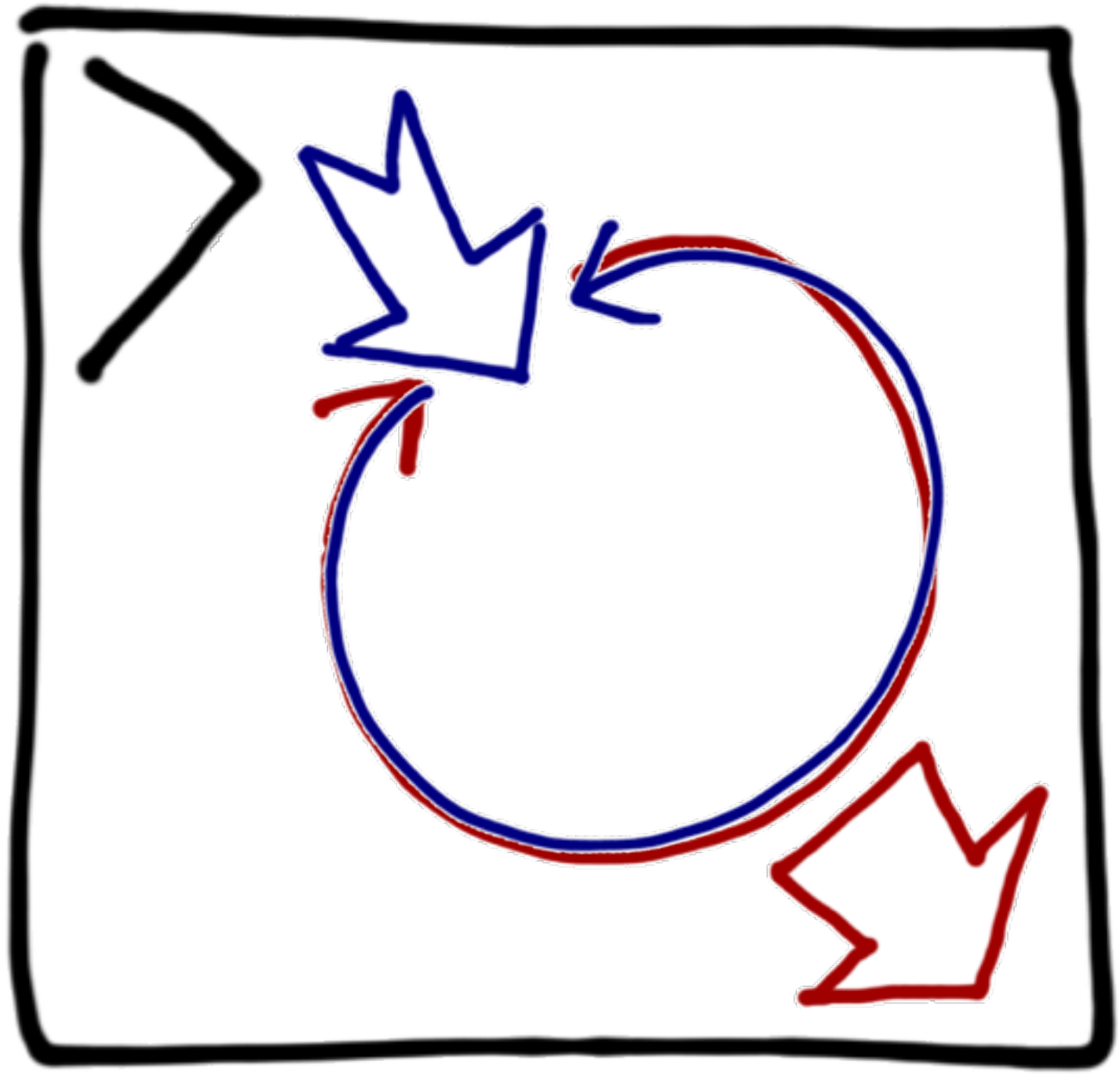
Um outro laço lê os registros a converter



Implementação possível: o mesmo laço lê e grava



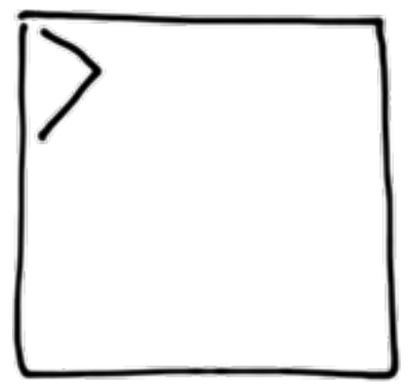
Mas e a lógica para ler outro formato?



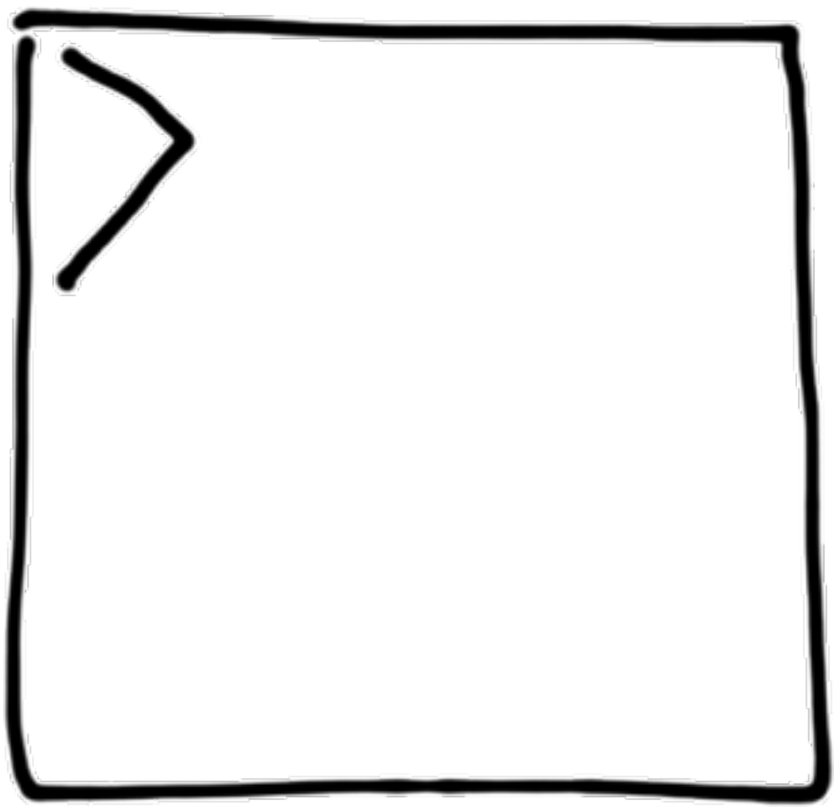
Funções do script

- iterMstRecords*
- iterIsoRecords*
- writeJsonArray
- main

* funções geradoras



Função main: leitura dos argumentos



```python
def main():
    # create the parser
    parser = argparse.ArgumentParser(
        description='Convert an ISIS .mst or .iso file to a JSON array')

    # add the arguments
    parser.add_argument(
        'file_name', metavar='INPUT.(mst|iso)',
        help='.mst or .iso file to read')
    parser.add_argument(
        '-o', '--out', type=argparse.FileType('w'), default=sys.stdout,
        metavar='OUTPUT.json',
        help='the file where the JSON output should be written'
             ' (default: write to stdout)')
    parser.add_argument(
        '-c', '--couch', action='store_true',
        help='output array within a "docs" item in a JSON document'
             ' for bulk insert to CouchDB via POST to db/_bulk_docs')
    parser.add_argument(
        '-m', '--mongo', action='store_true',
        help='output individual records as separate JSON dictionaries,'
             ' one per line for bulk insert to MongoDB via mongoimport utility')
    parser.add_argument(
        '-t', '--type', type=int, metavar='ISIS_JSON_TYPE', default=1,
        help='ISIS-JSON type, sets field structure: 1=string, 2=alist, 3=dict '
             '(default=1)')
    parser.add_argument(
        '-q', '--qty', type=int, default=DEFAULT_QTY,
        help='maximum quantity of records to read (default=ALL)')
    parser.add_argument(
        '-s', '--skip', type=int, default=0,
        help='records to skip from start of .mst (default=0)')
    parser.add_argument(
        '-i', '--id', type=int, metavar='TAG_NUMBER', default=0,
        help='generate an "_id" from the given unique TAG field number'
             ' for each record')
    parser.add_argument(
        '-u', '--uuid', action='store_true',
        help='generate an "_id" with a random UUID for each record')
    parser.add_argument(
        '-p', '--prefix', type=str, metavar='PREFIX', default='',
        help='concatenate prefix to every numeric field tag '
             '(ex. 99 becomes "v99")')
    parser.add_argument(
        '-n', '--mfn', action='store_true',
        help='generate an "_id" from the MFN of each record'
             ' (available only for .mst input)')
    parser.add_argument(
        '-k', '--constant', type=str, metavar='TAG:VALUE', default='',
        help='Include a constant tag:value in every record (ex. -k type:AS)')

    '''
    # TODO: implement this to export large quantities of records to CouchDB
    parser.add_argument(
        '-r', '--repeat', type=int, default=1,
        help='repeat operation, saving multiple JSON files'
             ' (default=1, use -r 0 to repeat until end of input)')
    '''
    # parse the command line
    args = parser.parse_args()
    if args.file_name.lower().endswith('.mst'):
        iterRecords = iterMstRecords
    else:
        if args.mfn:
            print('UNSUPORTED: -n/--mfn option only available for .mst input.')
            raise SystemExit
        iterRecords = iterIsoRecords
    if args.couch:
        args.out.write('{ "docs" : ')
    writeJsonArray(iterRecords, args.file_name, args.out, args.qty, args.skip,
        args.id, args.uuid, args.mongo, args.mfn, args.type, args.prefix,
        args.constant)
    if args.couch:
        args.out.write('}\n')
    args.out.close()

if __name__ == '__main__':
    main()
```

Função main: seleção do formato de entrada

escolha da função geradora de leitura depende do formato de entrada

```
    args = parser.parse_args()
    if args.file_name.lower().endswith('.mst'):
        iterRecords = iterMstRecords
    else:
        if args.mfn:
            print('UNSUPORTED: -n/--mfn option only available for .mst input.')
            raise SystemExit
        iterRecords = iterIsoRecords
    if args.couch:
        args.out.write('{ "docs" : ')
    writeJsonArray(iterRecords, args.file_name, args.out, args.qty, args.skip,
        args.id, args.uuid, args.mongo, args.mfn, args.type, args.prefix,
        args.constant)
    if args.couch:
        args.out.write('}\n')
    args.out.close()

if __name__ == '__main__':
    main()
```

função geradora escolhida é passada como argumento

writeJsonArray:

escrever registros em JSON

```
def writeJsonArray(iterRecords, file_name, output, qty, skip, id_tag,
                   gen_uuid, mongo, mfn, isis_json_type, prefix, constant):
    start = skip
    end = start + qty
    if not mongo:
        output.write('[')
    if id_tag:
        id_tag = str(id_tag)
        ids = set()
    else:
        id_tag = ''
    for i, record in enumerate(iterRecords(file_name, isis_json_type)):
        if i >= end:
            break
        if i > start and not mongo:
            output.write(',')
        output.write('\n')
        if start <= i < end:
            if id_tag:
                occurrences = record.get(id_tag, None)
                if occurrences is None:
                    msg = 'id tag #%s not found in record %s'
                    if ISIS_MFN_KEY in record:
                        msg = msg + (' (mfn=%s)' % record[ISIS_MFN_KEY])
                    raise KeyError(msg % (id_tag, i))
                if len(occurrences) > 1:
                    msg = 'multiple id tags #%s found in record %s'
                    if ISIS_MFN_KEY in record:
                        msg = msg + (' (mfn=%s)' % record[ISIS_MFN_KEY])
                    raise TypeError(msg % (id_tag, i))
                else: # ok, we have one and only one id field
                    if isis_json_type == 1:
                        id = occurrences[0]
                    elif isis_json_type == 2:
                        id = occurrences[0][0][1]
                    elif isis_json_type == 3:
                        id = occurrences[0]['_']
                    if id in ids:
                        msg = 'duplicate id %s in tag #%s, record %s'
                        if ISIS_MFN_KEY in record:
                            msg = msg + (' (mfn=%s)' % record[ISIS_MFN_KEY])
                        raise TypeError(msg % (id, id_tag, i))
                    record['_id'] = id
                    ids.add(id)
            elif gen_uuid:
                record['_id'] = unicode(uuid4())
            elif mfn:
                record['_id'] = record[ISIS_MFN_KEY]
            if prefix:
                # iterate over a fixed sequence of tags
                for tag in tuple(record):
                    if str(tag).isdigit():
                        record[prefix+tag] = record[tag]
                        del record[tag] # this is why we iterate over a tuple
                        # with the tags, and not directly on the record dict
            if constant:
                constant_key, constant_value = constant.split(':')
                record[constant_key] = constant_value
            output.write(json.dumps(record).encode('utf-8'))
    if not mongo:
        output.write('\n]')
    output.write('\n')
```

Turing.com.br

@ramalhoorg

writeJsonArray:

itera sobre umas das funções geradoras

```
def writeJsonArray(iterRecords, file_name, output, qty, skip, id_tag,
                   gen_uuid, mongo, mfn, isis_json_type, prefix, constant):
    start = skip
    end = start + qty
    if not mongo:
        output.write('[')
    if id_tag:
        id_tag = str(id_tag)
        ids = set()
    else:
        id_tag = ''
    for i, record in enumerate(iterRecords(file_name, isis_json_type)):
        if i >= end:
            break
        if i > start and not mongo:
            output.write(',')
        output.write('\n')
        if start <= i < end:
            if id_tag:
```

iterIsoRecords:

ler registros de arquivo ISO-2709

função geradora!

```
def iterIsoRecords(iso_file_name, isis_json_type):
    from iso2709 import IsoFile
    from subfield import expand

    iso = IsoFile(iso_file_name)
    for record in iso:
        fields = {}
        for field in record.directory:
            field_key = str(int(field.tag)) # remove leading zeroes
            field_occurrences = fields.setdefault(field_key,[])
            content = field.value.decode(INPUT_ENCODING,'replace')
            if isis_json_type == 1:
                field_occurrences.append(content)
            elif isis_json_type == 2:
                field_occurrences.append(expand(content))
            elif isis_json_type == 3:
                field_occurrences.append(dict(expand(content)))
            else:
                raise NotImplementedError(
                    'ISIS-JSON type %s conversion not yet '
                    'implemented for .iso input' % isis_json_type)

        yield fields
    iso.close()
```

Turing.com.br

@ramalhoorg

iterIsoRecords

```
def iterIsoRecords(iso_file_name, isis_json_type):
    from iso2709 import IsoFile
    from subfield import expand

    iso = IsoFile(iso_file_name)
    for record in iso:
        fields = {}
        for field in record.directory:
            field_key = str(int(field.tag)) # remove leading zeroes
            field_occurrences = fields.setdefault(field_key,[])
            content = field.value.decode(INPUT_ENCODING,'replace')
            if isis_json_type == 1:
                field_occurrences.append(content)
            elif isis_json_type == 2:
                field_occurrences.append(expand(content))
            elif isis_json_type == 3:
                field_occurrences.append(dict(expand(content)))
            else:
                raise NotImplementedError(
                    'ISIS-JSON type %s conversion not yet '
                    'implemented for .iso input' % isis_json_type)

        yield fields
    iso.close()
```

cria um novo dict a cada iteração

produz (yield) registro na forma de um dict

iterMstRecords:

função geradora!

ler registros de arquivo ISIS .MST

```python
def iterMstRecords(master_file_name, isis_json_type):
    try:
        from bruma.master import MasterFactory, Record
    except ImportError:
        print('IMPORT ERROR: Jython 2.5 and Bruma.jar are required '
              'to read .mst files')
        raise SystemExit
    mst = MasterFactory.getInstance(master_file_name).open()
    for record in mst:
        fields = {}
        if SKIP_INACTIVE:
            if record.getStatus() != Record.Status.ACTIVE:
                continue
        else: # save status only there are non-active records
            fields[ISIS_ACTIVE_KEY] = record.getStatus() == Record.Status.ACTIVE
        fields[ISIS_MFN_KEY] = record.getMfn()
        for field in record.getFields():
            field_key = str(field.getId())
            field_occurrences = fields.setdefault(field_key,[])
            if isis_json_type == 3:
                content = {}
                for subfield in field.getSubfields():
                    subfield_key = subfield.getId()
                    if subfield_key == '*':
                        content['_'] = subfield.getContent()
                    else:
                        subfield_occurrences = content.setdefault(subfield_key,[])
                        subfield_occurrences.append(subfield.getContent())
                field_occurrences.append(content)
            elif isis_json_type == 1:
                content = []
                for subfield in field.getSubfields():
                    subfield_key = subfield.getId()
                    if subfield_key == '*':
                        content.insert(0, subfield.getContent())
                    else:
                        content.append(SUBFIELD_DELIMITER+subfield_key+
                                       subfield.getContent())
                field_occurrences.append(''.join(content))
            else:
                raise NotImplementedError(
                    'ISIS-JSON type %s conversion not yet '
                    'implemented for .mst input' % isis_json_type)
        yield fields
    mst.close()
```

Turing.com.br

@ramalhoorg

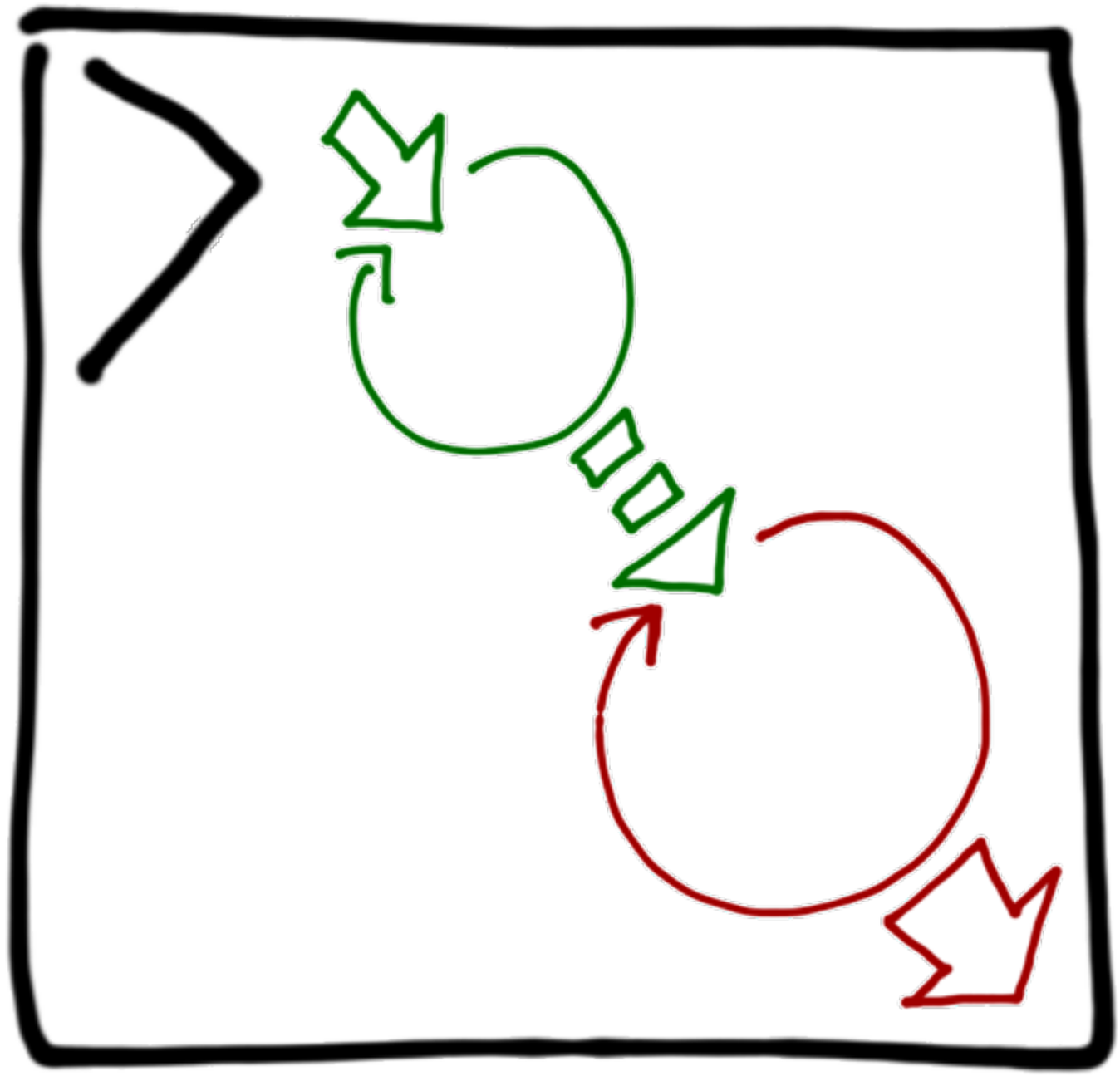
iterMstRecords

```
def iterMstRecords(master_file_name, isis_json_type):
    try:
        from bruma.master import MasterFactory, Record
    except ImportError:
        print('IMPORT ERROR: Python 2.x + Bruma.jar are required '
              'to read .mst files')
        raise SystemExit
    mst = MasterFactory.getInstance(master_file_name).open()
    for record in mst:
        fields = {}
        if SKIP_INACTIVE:
            if record.getStatus() != Record.Status.ACTIVE:
                continue
        else: # save status only there are non-active records
            fields[ISIS_ACTIVE_KEY] = record.getStatus() == Record.Status.ACTIVE
        fields[ISIS_MFN_KEY] = record.getMfn()
        for field in record.getFields():
            field_key = str(field.getId())
            field_occurrences = fields.setdefault(field_key,[])
            if isis_json_type == 3:
                content = {}
                for subfield in field.getSubfields():
                    subfield_key = subfield.getId()
                    if subfield_key == '*':
                        content['_'] = subfield.getContent()
                    else:
                        subfield_occurrences = content.setdefault(subfield_key,[])
                        subfield_occurrences.append(subfield.getContent())
                field_occurrences.append(content)
            elif isis_json_type == 1:
                content = []
                for subfield in field.getSubfields():
                    subfield_key = subfield.getId()
                    if subfield_key == '*':
                        content.insert(0, subfield.getContent())
                    else:
                        content.append(SUBFIELD_DELIMITER+subfield_key+
                                       subfield.getContent())
                field_occurrences.append(''.join(content))
            else:
                raise NotImplementedError(
                    'ISIS-JSON type %s conversion not yet '
                    'implemented for .mst input' % isis_json_
        yield fields
    mst.close()
```

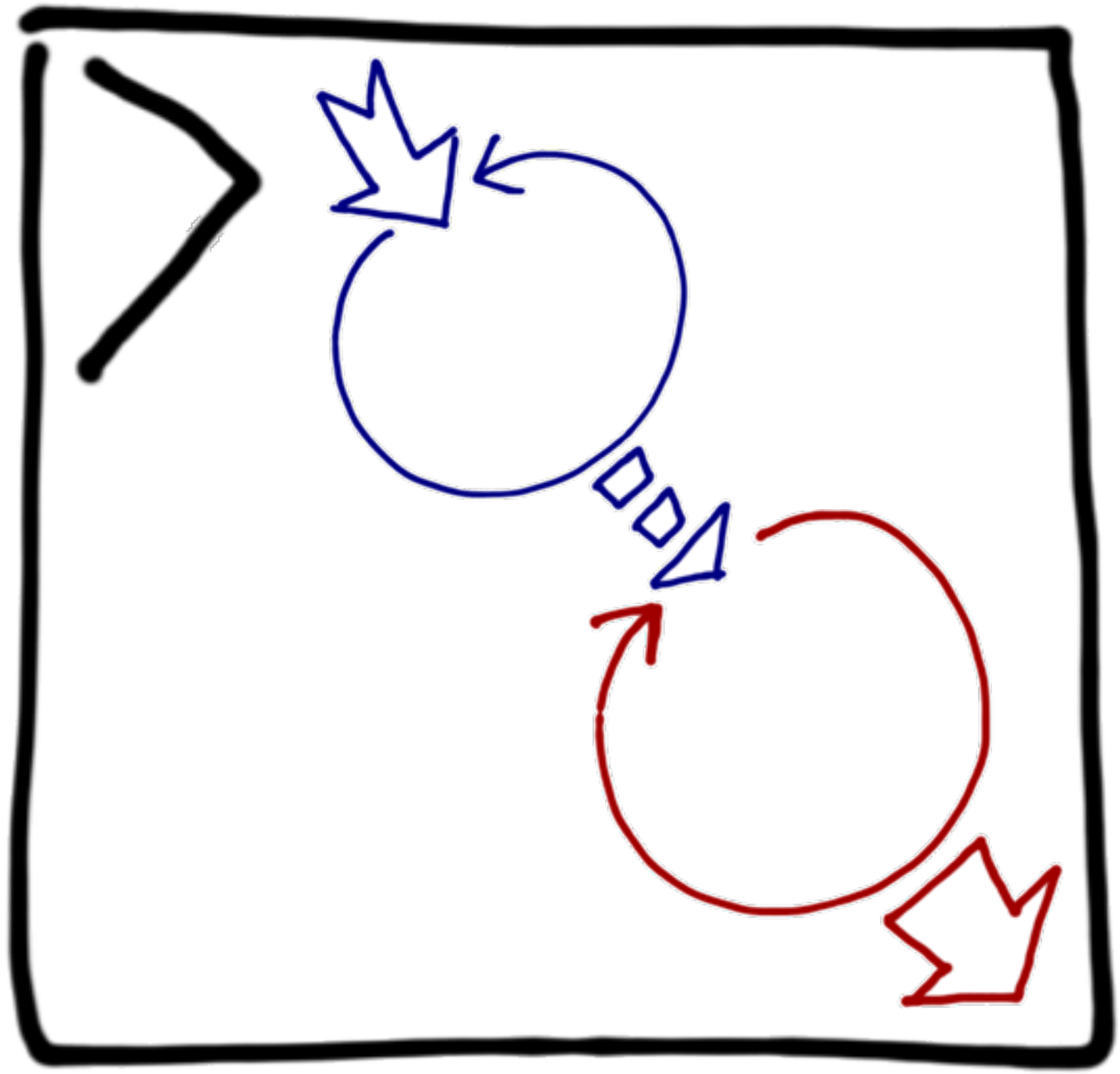
cria um novo dict
a cada iteração

produz (yield) registro
na forma de um dict

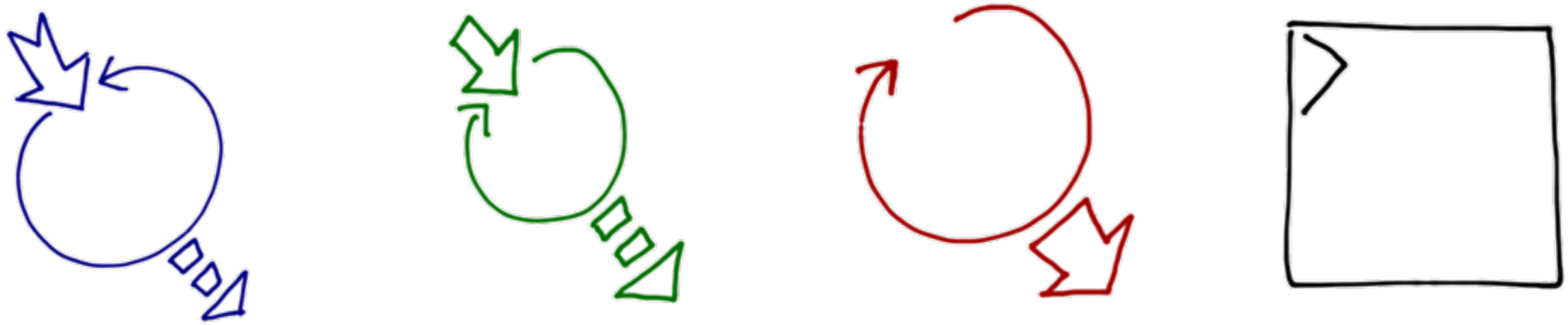
Geradores na prática



Geradores na prática



Geradores na prática



```
def itemGetRecords(master_file_name, isis_json_type):
    try:
        from bruma.master import MasterFactory, Record
    except ImportError:
        print('IMPORT ERROR: Jython 2.5 and Bruma.jar are required '
              'to read .mst files')
        raise SystemExit
    mst = MasterFactory.getInstance(master_file_name).open()
    for record in mst:
        fields = {}
        if DEBUG_INACTIVE:
            if record.getStatus() != Record.Status.ACTIVE:
                continue
        else: # save status only there are non-active records
            fields[ISIS_ACTIVE_KEY] = record.getStatus() == Record.Status.ACTIVE
        fields[ISIS_MFN_KEY] = record.getMfn()
        for field in record.getFields():
            field_key = str(field.getTag())
            field_occurrences = fields.setdefault(field_key,[])
            if isis_json_type == 1:
                content = {}
                for subfield in field.getSubfields():
                    subfield_key = subfield.getCode()
                    if subfield_key == '':
                        content['_'] = subfield.getContent()
                    else:
                        subfield_occurrence = content.setdefault(subfield_key,[])
                        subfield_occurrence.append(subfield.getContent())
                field_occurrence = content
            elif isis_json_type == 2:
                content = []
                for subfield in field.getSubfields():
                    subfield_key = subfield.getCode()
                    if subfield_key == '':
                        content.insert(0, subfield.getContent())
                    else:
                        content.append(SUBFIELD_DELIMITER+subfield_key+
                                       subfield.getContent())
                field_occurrence = ''.join(content)
            else:
                raise NotImplementedError(
                  'ISIS-JSON type %s conversion not yet '
                  'implemented for .mst input' % isis_json_type)
        yield fields
    mst.close()
```

```
def itemIsoRecords(iso_file_name, isis_json_type):
    from iso2709 import IsoFile
    from subfield import expand

    iso = IsoFile(iso_file_name)
    for record in iso:
        fields = {}
        for field in record.directory:
            field_key = str(int(field.tag)) # remove leading zeroes
            field_occurrences = fields.setdefault(field_key,[])
            content = field.value.decode(INPUT_ENCODING, 'replace')
            if isis_json_type == 1:
                field_occurrences.append(content)
            elif isis_json_type == 2:
                field_occurrences.append(expand(content))
            elif isis_json_type == 3:
                field_occurrences.append(dict(expand(content)))
            else:
                raise NotImplementedError(
                    'ISIS-JSON type %s conversion not yet '
                    'implemented for .iso input' % isis_json_type)
        yield fields
    iso.close()
```

```
def writeJsonArray(itemRecords, file_name, output, qty, skip, id_tag,
                   gen_uuid, mongo, mfn, isis_json_type, prefix, constant):
    start = skip
    end = start + qty
    if not mongo:
        output.write('[')
    if id_tag:
        id_tag = str(id_tag)
        ids = set()
    else:
        id_tag = ''
    for i, record in enumerate(itemRecords(file_name, isis_json_type)):
        if i >= end:
            break
        if i > start and not mongo:
            output.write(',')
        output.write('\n')
        if start <= i < end:
            if id_tag:
                occurrence = record.get(id_tag, None)
                if occurrence is None:
                    msg = 'id tag #%s not found in record %r'
                    if ISIS_MFN_KEY in record:
                        msg = msg + ' (mfn=%r)' % record[ISIS_MFN_KEY])
                    raise KeyError(msg % (id_tag, i))
                if len(occurrence) > 1:
                    msg = 'multiple id tags #%s found in record %r'
                    if ISIS_MFN_KEY in record:
                        msg = msg + ' (mfn=%r)' % record[ISIS_MFN_KEY])
                    raise TypeError(msg % (id_tag, i))
                else: # ok, we have one and only one id field
                    if isis_json_type == 1:
                        id = occurrence[0]
                    elif isis_json_type == 2:
                        id = occurrence[0][1:]
                    elif isis_json_type == 3:
                        id = occurrence[0].get('_')
                    if id in ids:
                        msg = 'duplicate id %r in tag #%s, record %r'
                        if ISIS_MFN_KEY in record:
                            msg = msg + ' (mfn=%r)' % record[ISIS_MFN_KEY])
                        raise TypeError(msg % (id, id_tag, i))
                    record['_id'] = id
                    ids.add(id)
            elif gen_uuid:
                record['_id'] = unicode(uuid4())
            elif mfn:
                record['_id'] = record[ISIS_MFN_KEY]
            if prefix:
                # iterate over a fixed sequence of tags
                for tag in tuple(record):
                    if str(tag).isdigit():
                        record[prefix+tag] = record[tag]
                        del record[tag] # this is why we iterate over a tuple
                                        # with the tags, and not directly on the record dict
            if constant:
                constant_key, constant_value = constant.split(':')
                record[constant_key] = constant_value
            output.write(json.dumps(record).encode('utf-8'))
    if not mongo:
        output.write('\n]')
    output.write('\n')
```

```
def main():
    # create the parser
    parser = argparse.ArgumentParser(
        description='Convert an ISIS .mst or .iso file to a JSON array')

    # add the arguments
    parser.add_argument(
        'file_name', metavar='INPUT.{mst|iso}',
        help='.mst or .iso file to read')
    parser.add_argument(
        'output', type=argparse.FileType('w'), default=sys.stdout,
        metavar='OUTPUT.json',
        help='the file where the JSON output should be written'
             ' (default: write to stdout)')
    parser.add_argument(
        '-c', '--couch', action='store_true',
        help='output array within a "docs" item in a JSON document'
             ' for bulk insert to CouchDB via POST to db/_bulk_docs')
    parser.add_argument(
        '-m', '--mongo', action='store_true',
        help='output individual records as separate JSON dictionaries,'
             ' one per line for bulk insert to MongoDB via mongoimport utility')
    parser.add_argument(
        '-t', '--type', type=int, metavar='ISIS_JSON_TYPE', default=1,
        help='ISIS-JSON type, sets field structure: 1=string, 2=value, 3=dict '
             '(default=1)')
    parser.add_argument(
        '-q', '--qty', type=int, default=DEFAULT_QTY,
        help='maximum quantity of records to read (default=ALL)')
    parser.add_argument(
        '-s', '--skip', type=int, default=0,
        help='records to skip from start of .mst (default=0)')
    parser.add_argument(
        '-i', '--id', type=int, metavar='TAG_NUMBER', default=0,
        help='generate an "_id" from the given unique TAG field number'
             ' for each record')
    parser.add_argument(
        '-u', '--uuid', action='store_true',
        help='generate an "_id" with a random UUID for each record')
    parser.add_argument(
        '-p', '--prefix', type=str, metavar='PREFIX', default='',
        help='concatenate prefix to every numeric field tag '
             '(ex. "v" becomes "v99")')
    parser.add_argument(
        '-n', '--mfn', action='store_true',
        help='generate an "_id" from the MFN of each record'
             ' (available only for .mst input)')
    parser.add_argument(
        '-k', '--constant', type=str, metavar='TAG:VALUE', default='',
        help='include a constant tag:value in every record (ex. -k type:AS)')
    ...
    # TODO: implement this to export large quantities of records to CouchDB
    parser.add_argument(
        '-r', '--repeat', type=int, default=1,
        help='repeat operation, saving multiple JSON files'
             ' (default=1, use -r 0 to repeat until end of input)')
    ...
    # parse the command line
    args = parser.parse_args()
    if args.file_name.lower().endswith('.mst'):
        itemRecords = itemMstRecords
    else:
        if args.mfn:
            print('UNSUPPORTED: -n/--mfn option only available for .mst input.')
            raise SystemExit
        itemRecords = itemIsoRecords
    if args.couch:
        args.out.write('{ "docs" : ')
    writeJsonArray(itemRecords, args.file_name, args.out, args.qty, args.skip,
        args.id, args.uuid, args.mongo, args.mfn, args.type, args.prefix,
        args.constant)
    if args.couch:
        args.out.write('}\n')
    args.out.close()

if __name__ == '__main__':
    main()
```

Turing.com.br

Encapsulamento

Encapsulamento

- Propriedades:
  - encapsulamento para quem precisa de encapsulamento

```
>>> a = C()
>>> a.x = 10
>>> print a.x
10
>>> a.x = -10
>>> print a.x
0
```

violação de encapsulamento?

pergunte-me como!

slide semi novo

Propriedade: implementação

- apenas para leitura, via decorator:

```
class C(object):
    def __init__(self, x):
        self.__x = x
    @property
    def x(self):
        return self.__x
```

- a notação __x protege o atributo contra acessos acidentais (__x = dois underscores à esquerda)

slide semi novo

Propriedade: implementação 2

- para leitura e escrita (Python >= 2.2):

```
class C(object):
    def __init__(self, x=0):
        self.__x = x
    def getx(self):
        return self.__x
    def setx(self, valor):
        self.__x = valor if valor >= 0 else 0
    x = property(getx, setx)
```

slide
semi
novo

Propriedade: implementação 3

- para leitura e escrita (Python >= 2.6):

```
class C(object):
    def __init__(self, x=0):
        self.__x = x
    @property
    def x(self):
        return self.__x
    @x.setter
    def x(self, valor):
        self.__x = valor if valor >= 0 else 0
```

slide
semi
novo

Propriedade: exemplo de uso

```
class ContadorTotalizador(Contador):
    def __init__(self):
        super(ContadorTotalizador, self).__init__()
        self.__total = 0

    def incluir(self, item):
        super(ContadorTotalizador, self).incluir(item)
        self.__total += 1

    @property
    def total(self):
        return self.__total
```

slide
semi
novo

Atributos protegidos

- Atributos protegidos em Python são salvaguardas
  - servem para evitar atribuição ou sobrescrita acidental
  - não para evitar usos (ou abusos) intencionais

Atributos protegidos

- Atributos protegidos em Python são salvaguardas
  - servem para evita atribuição ou sobrescrita acidental
  - não para evitar usos (ou abusos) intencionais

```
>>> raisins._LineItem__weight
10
```

Decoradores

Decoradores de funções

- Exemplos de decoradores:
  - @property, @x.setter, @classmethod
- Não têm relação com o padrão de projeto "decorator"
- São funções que recebem a função decorada como argumento e produzem uma nova função que substitui a função decorada
  - Tema de outro curso...

Decoradores de métodos

- Usados na definição de métodos em classes
  - @property, @x.setter, @x.deleter: definem métodos getter, setter e deleter para propriedades
  - @classmethod, @staticmethod: definem métodos que não precisam de uma instância para operar
  - @abstractmethod, @abstractproperty: uso em classes abstratas (veremos logo mais)

classmethod x staticmethod

- Métodos estáticos são como funções simples embutidas em uma classe: não recebem argumentos automáticos
- Métodos de classe recebem a classe como argumento automático

```
class Exemplo(object):
    @staticmethod
    def estatico(arg):
        return arg
    @classmethod
    def da_classe(cls, arg):
        return (cls, arg)
```

```
>>> Exemplo.estatico('bar')
'bar'
>>> Exemplo.da_classe('fu')
(<class '__main__.Exemplo'>, 'fu')
```

Carta simples

```
class Carta(object):

    def __init__(self, valor, naipe):
        self.valor = valor
        self.naipe = naipe

    def __repr__(self):
        return 'Carta(%r, %r)' % (self.valor, self.naipe)
```

```
>>> zape = Carta('4', 'paus')
>>> zape.valor
'4'
>>> zape
Carta('4', 'paus')
```

Todas as cartas

```
class Carta(object):

    naipes = 'espadas ouros paus copas'.split()
    valores = '2 3 4 5 6 7 8 9 10 J Q K A'.split()

    def __init__(self, valor, naipe):
        self.valor = valor
        self.naipe = naipe

    def __repr__(self):
        return 'Carta(%r, %r)' % (self.valor, self.naipe)

    @classmethod
    def todas(cls):
        return [cls(v, n) for n in cls.naipes
                          for v in cls.valores]
```

```
>>> monte = Carta.todas()
>>> len(monte)
52
>>> monte[0]
Carta('2', 'espadas')
>>> monte[-3:]
[Carta('Q', 'copas'), Carta('K', 'copas'), Carta('A', 'copas')]
```

Exemplo de classmethod

- É conveniente em **todas** ter acesso à classe para acessar os atributos (**naipes**, **valores**) e para instanciar as cartas

```
class Carta(object):

    naipes = 'espadas ouros paus copas'.split()
    valores = '2 3 4 5 6 7 8 9 10 J Q K A'.split()

    def __init__(self, valor, naipe):
        self.valor = valor
        self.naipe = naipe

    def __repr__(self):
        return 'Carta(%r, %r)' % (self.valor, self.naipe)

    @classmethod
    def todas(cls):
        return [cls(v, n) for n in cls.naipes
                          for v in cls.valores]
```

Decoradores de classes

- Novidade do Python 2.6, ainda pouco utilizada na prática
- Exemplo na biblioteca padrão a partir do Python 2.7:
  - **functools.total_ordering** define automaticamente métodos para os operadores de comparação **< > <= >=**

http://docs.python.org/library/functools.html#functools.total_ordering

Cartas comparáveis

```
from functools import total_ordering

@total_ordering
class Carta(object):

    naipes = 'espadas ouros paus copas'.split()
    valores = '2 3 4 5 6 7 8 9 10 J Q K A'.split()

    def __init__(self, valor, naipe):
        self.valor = valor
        self.naipe = naipe

    def __repr__(self):
        return 'Carta(%r, %r)' % (self.valor, self.naipe)

    @classmethod
    def todas(cls):
        return [cls(v, n) for n in cls.naipes
                          for v in cls.valores]

    def __eq__(self, outra):
        return ((self.valor, self.naipe) ==
                (outra.valor, outra.naipe))

    def peso(self):
        return (Carta.naipes.index(self.naipe) +
            len(Carta.naipes) * Carta.valores.index(self.valor))

    def __gt__(self, outra):
        return self.peso() > outra.peso()
```

```
>>> dois >= as_
False
>>> dois <= as_
True
```

Total ordering == lucro!

```
>>> mao = [as_, dois, rei]
>>> sorted(mao)
[Carta('2', 'espadas'), Carta('K', 'copas'), Carta('A', 'copas')]
```

```python
from functools import total_ordering

@total_ordering
class Carta(object):

    # ...várias linhas omitidas...

    def __eq__(self, outra):
        return ((self.valor, self.naipe) ==
                (outra.valor, outra.naipe))

    def peso(self):
        return (Carta.naipes.index(self.naipe) +
            len(Carta.naipes) * Carta.valores.index(self.valor))

    def __gt__(self, outra):
        return self.peso() > outra.peso()
```

Classes abstratas

Abstract Base Classes

- Novidade a partir do Python 2.6
  - a linguagem fez sucesso por 20 anos sem isso
- Permite pela primeira vez marcar explicitamente uma classe como abstrata e também métodos abstratos
- Documentação: procure “Abstract Base Classes” http://docs.python.org/2/library/abc.html

Abstract Base Classes

- Em Python (e C++ e outras linguagens), classes abstratas são usadas para definir bases comuns e para formalizar interfaces
- Por ser um recurso recente da linguagem, poucos projetos usam classes abstratas formalmente
  - mas muitas classes abstratas já existem informalmente
- Exemplos: collections ABC (Library Reference)

Cao abstrato

```
from abc import ABCMeta, abstractmethod

class Cao(object):
    __metaclass__ = ABCMeta

    qt_patas = 4
    carnivoro = True

    def __init__(self, nome):
        self.nome = nome

    @abstractmethod
    def latir(self, vezes=1):
        """ deve exibir o latido no stdout """

    def __str__(self):
        return self.nome

    def __repr__(self):
        return '{0}({1!r})'.format(self.__class__.__name__, self.nome)
```

Cao abstrato

```
$ python -i cao_abc.py
>>> Cao
<class '__main__.Cao'>
>>> Cao.__metaclass__
<class 'abc.ABCMeta'>
>>> class Viralata(Cao):
...     """um cão genérico"""
...
>>> pipo = Viralata('Pipoe')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class Viralata with
abstract methods latir
```

Cao concreto

```
class Mastiff(Cao):
    """ O mastiff late diferente:

        >>> atos = Mastiff('Atos')
        >>> atos.latir()
        Atos: Wuff!
    """
    def latir(self, vezes=1):
        # o mastiff não muda seu latido quando nervoso
        print self.nome + ':' + ' Wuff!' * vezes
```

```
>>> adamastor = Mastiff('Adamastor')
>>> adamastor.latir()
Adamastor: Wuff!
```