Python Mágico

Autores:
Laura Victoria Riera Pérez
Leandro Rodríguez Llosa
Marcos Manuel Tirador del Riego

Grupo: C-311

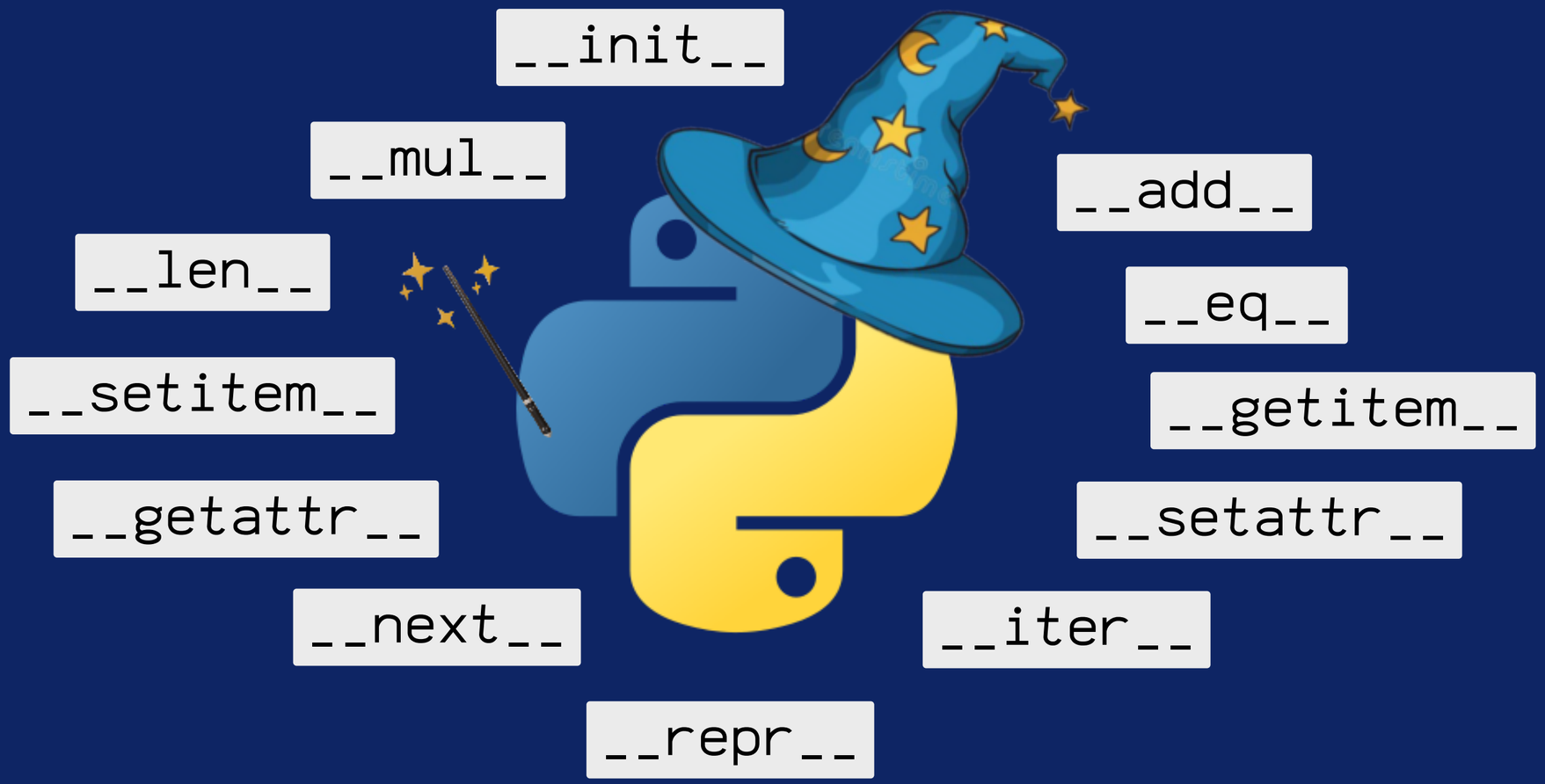
Facultad de Matemática y Computación
Universidad de La Habana, Cuba

Noviembre, 2022

Índice general

Métodos mágicos

Conjunto de métodos especiales en Python, cuyo nombre comienza y termina con dos guiones bajos. Son invocados internamente desde la clase, bajo una determinada acción.


__init__
__mul__
__len__
__setitem__
__getattr__
__next__
__repr__
__add__
__eq__
__getitem__
__setattr__
__iter__

`__init__`

Es invocado cuando se crea una
instancia de una clase, actúa como un constructor.

```python
def __init__(self, rows: int, cols: int, init_value=None):
    """
        Inicializador de la clase.
    """
    self.amount_rows: int = rows
    self.amount_cols: int = cols
    self.matrix: List[List[T]] = [[init_value
                                   for _ in range(0, cols)]
                                  for _ in range(0, rows)]
```

Operadores

Los métodos mágicos permiten
realizar operaciones
con los objetos definidos como si
fueran built-in.

`__eq__`

Permite comparar dos objetos de un mismo tipo haciendo uso del operador `==`.

```python
def __eq__(self, other: 'Matrix[T]') -> bool:
    """
        Redeinición del operador `==`, para poder construir
        expresiones de la forma:

        `if matrix1 == matrix2: pass`.
    """
    if self.amount_rows != other.amount_rows or \
        self.amount_cols != other.amount_cols:
        return False

    for i,j in zip(self,other):
        if i != j:
            return False
    return True
```

`__add__`

Permite sumar dos objetos de un mismo tipo haciendo uso del operador `+`.

```python
def __add__(self, other: 'Matrix[T]') -> 'Matrix[T]':
    """
        Sumador de matrices.
    """
    result = Matrix[T](rows=self.amount_rows,
        cols=self.amount_cols, init_value=None)

    for i in range(0, self.amount_rows):
        for j in range(0, self.amount_cols):
            result[i, j] = self[i, j] + other[i, j]

    return result
```

`__mul__`

Permite multiplicar dos objetos de un mismo tipo haciendo uso del operador `*` .

```python
def __mul__(self, other: 'Matrix[T]') -> 'Matrix[T]':
    """
        Multiplicador de matrices.
    """
    if self.amount_cols != other.amount_rows:
        raise Exception('Invalid operation.')

    result = Matrix[T](rows=self.amount_rows,
        cols=other.amount_cols, init_value=None)

    if self.amount_cols != other.amount_rows:
        raise Exception('Invalid operation.')

    result = Matrix(rows=self.amount_rows,
                    cols=other.amount_cols, init_value=None)

    for i in range(0, self.amount_rows):
        for j in range(0, other.amount_cols):
            for h in range(0, self.amount_cols):
                if h == 0:
                    result[i, j] = self[i, h] * other[h, j]
                else:
                    result[i, j] += self[i, h] * other[h, j]

    return result
```

Indexación

Los métodos mágicos también permiten añadir la funcionalidad de indexar mediante corchetes para acceder al valor de un elemento.

`__getitem__`

Permite leer el valor de un elemento mediante el uso de corchetes.

```python
def __getitem__(self, key: Tuple[int, int]):
    """
        Indizador con una sintaxis más cómoda.

        Ejemplo: a = matrix[i,j]
    """
    if not isinstance(key, tuple):
        raise Exception('Format incorrect.')

    if len(key) != 2:
        raise Exception('Number of parameters exceded.')
    i, j = key

    if i >= 0 and i < self.amount_rows and \
        j >= 0 and j < self.amount_cols:
        return self.matrix[i][j]
    else:
        raise Exception('Index out of matrix.')
```

`__setitem__`

Permite cambiar el valor de un elemento mediante el uso de corchetes.

```python
def __setitem__(self, key: Tuple[int, int], value: T):
    """
        Permite setear el valor de la matriz indexada,
        de manera más cómoda.

        Ejemplo: `matrix[i,j] = 4`
    """
    if not isinstance(key, tuple):
        raise Exception('Format incorrect.')

    if len(key) != 2:
        raise Exception('Number of parameters exceded.')
    i, j = key

    if i >= 0 and i < self.amount_rows and \
        j >= 0 and j < self.amount_cols:
        self.matrix[i][j] = value
    else:
        raise Exception('Index out of matrix.')
```

Iteración

En Python un **iterable** es un objeto sobre el que se puede iterar, mientras que un **iterador** es un objeto que se usa para iterar sobre un iterable. Cada iterador es también iterable, pero no todos los iterables son iteradores.

`__next__`

Devuelve el siguiente elemento de un iterador.

```
def __next__(self):
    """
        Método que se encarga de generar el próximo elemento de la
        colección, de manera `lazy`.
    """
    for row in self.matrix:
        for i in row:
            yield i
```

`__iter__`

Permite crear un iterador a partir de un iterable.

```
def __iter__(self):
    """
        Método que da una forma de iterar por los elementos de una
        colección. Es quien permite hacer construcciones del lenguaje
        de este estilo:

        `for item in matrix: print(item)`
    """
    return self.__next__()
```

Atributos

Además, los métodos mágicos
permiten definir un
comportamiento determinado
cuando se intente acceder
a un atributo de la clase, incluso
cuando este atributo no existe.

`__getattribute__`

Es invocado cuando se intenta acceder a un atributo de una clase,
sin importar si este existe o no

```python
def __getattribute__(self, __name: str) -> Any:
    return super().__getattribute__(__name)
```

```python
def __getattr__(self, __name: str):
    """
        Controla la petición de atributos que pertencen a una
        instancia de la clase y no se encuentran inicializados.

        Ejemplo:
        - `a = matrix._0_2`
        - `b = matrix.as_float()`
    """
    matched = re.match(r"_(\d+)_(\d+)", __name)
    if matched:
        i, j = matched.groups()
        i = int(i); j = int(j)
        return self[i, j]

    matched = re.match(r"as_([a-z]+)", __name)
    if matched:
        type = matched.groups()[0]
        result = Matrix(self.amount_rows, self.amount_cols)

        for i in range(0, self.amount_rows):
            for j in range(0, self.amount_cols):
                result[i, j] = eval(f'{type}(self.matrix[i][j])')

        return lambda: result
```

`__getattr__`

Es invocado siempre que se intente acceder a un atributo que no existe en un objeto.

`__setattr__`

Es invocado siempre que se intenta cambiar el valor de un atributo.

```python
def __setattr__(self, __name: str, __value: Any):
    """
        Setea en los atributos de la
    """
    matched = re.match(r"_(\d+)_(\d+)", __name)
    if matched:
        i, j = matched.groups()
        i = int(i); j = int(j)
        self[i, j] = __value
        return

    return super().__setattr__(__name, __value)
```

Otros métodos mágicos utilizados

```python
def __repr__(self) -> str:
    """
        Método para imprimir la matriz de la forma: `print(matrix)`.
    """
    result = ''
    for row in self.matrix:
        result += ''.join(str(row)) + '\n'
    return result
```

`__repr__`

Devuelve el siguiente elemento de un iterador.

```python
def __len__(self) -> int:
    """
        Método para saber el tamaño de una matriz.

        Ejemplo:
        - `len(matrix)`
    """
    return self.amount_cols * self.amount_rows
```

`__len__`

Permite crear un iterador a partir de un iterable.

Código y documentación

Para ver el funcionamiento de los métodos mágicos mostrados, ir a **matrix.py**

Para una mejor explicación de los métodos mágicos en Python, ir a **magic_python.pdf**

Ambos en el repositorio de github:
https://github.com/science-engineering-art/programming-languages

Gracias