JDOG

Release 1.0.0

Petr Nymsa

Jan 11, 2020

CONTENTS

CHAPTER

ONE

PLACEHOLDERS

Each placeholder serves as a place where the new, random value will be replaced in the output.

Generally each placeholder is enclosed between **{{** and **}}**. So for example:

```
{{placeholder}}
```

Some placeholders accept arguments and some arguments can be required or optional. See below details of each placeholder.

Optional placeholder within the table is denoted by *[* arg *]*.

Placeholder	Description
name([m,f])	Generic full name
first_name([m,f])	Generic first name
last_name([m,f])	Generic last name
number(l,h)	Number between l (inclusive) to h (exclusive)
age	Number between 1 and 99. Equal to number(1,100)
city	Generic city name
street_address	Generic street address
lorem(n)	Generic lorem ipsum text contaning n words
empty	Empty string field
range(prop,n,[m])	Generates property with "prop" name and array as its value
bool	Boolean value - true / false
option(arg1,arg2,...,argN)	Choose randomly one of the argument.

1.1 Defined placeholders

Follows a description of each defined placeholder available to use.

Note: Internally JDOG is using amazing Faker package to generate random values.

1.1.1 name([m,f])

Generic person full name - that is the first and last name.

Arguments

Optional m - Generates male names.

Optional f - Generates female names.

If none of these arguments is provided then generates a male or female name.

Example

```
{
    "full_name": "{{name}}"
}

# Example output

{
    "full_name": "Joe Hill"
}
```

1.1.2 first_name([m,f])

Generic person first name.

Arguments

Optional m - Generates male names.

Optional f - Generates female names.

If none of these arguments is provided then generates a male or female name.

Example

```
{
    "first": "{{first_name(m)}}"
}

# Example output

{
    "first": "Joe"
}
```

1.1.3 last_name([m,f])

Generic person last name.

Arguments

Optional m - Generates male names.

Optional f - Generates female names.

If none of these arguments is provided then generates a male or female name.

Example

```
{
    "last": "{{last_name(f)}}"
}

# Example output

{
    "last": "Hills"
}
```

1.1.4 number(l,h)

Generates number between *l* and *h*. Note that *h* is **exclusive**.

Arguments

- *l* - left boundary, inclusive
- *h* - right boundary, exclusive

Example

```
{
    "age": "{{number(1,100)}}"
}

# Example output

{
    "age": "42"
}
```

1.1.5 age

A random number from 1 to 99. Effectively the same as using *{{number(1,100)}}*.

Arguments

None.

Example

```
{
    "age": "{{age}}"
}

# Example output

{
    "age": "42"
}
```

1.1.6 city

City name.

Arguments

None

Example

```
{
    "born_city": "{{city}}"
}

# Example output

{
    "born_city": "Coruscant"
}
```

1.1.7 street_address

Generic street address.

Arguments

None.

Example

```
{
    "company_address": "{{street_address}}"
}

# Example output

{
    "company_address": "5th avenue"
}
```

1.1.8 lorem(n)

Random text containing *n* words.

Arguments

n - How many words should text contain.

Example

```
{
    "description": "{{lorem(6)}}"
}

# Example output

{
    "description": "Find control party plan water prove safe."
}
```

1.1.9 empty

Empty value. Useful with combination with *option* placeholder.

Arguments

None.

Example

```
{
    "title": "{{empty}}"
}

# Example output

{
    "title": ""
}
```

1.1.10 range(prop,n,[m])

Generates property named *prop* with an array of values. The number of benefits depends on arguments *n* and *m*.

Note that range placeholder *should be used at the left side* of property. See examples below.

Arguments

- *prop* - Name of property.
- *n* - If only *n* specified array contains exactly *n* values.
- **optional** *m* - If *m* is specified array contains items exactly between *n* up to *m* times.

Example

Generate exactly four people objects.

```
{
  "{{range(people,4)}}": {
    "name": "{{name}}",
    "age": "{{age}}",
    "address": {
      "city": "{{city}}"
    },
    "car": "{{option(mustang,{{empty}})}}"
  }
}

# Example output

{
    "people": [
        {
            "name": "Brandi Young",
            "age": 39,
            "address": {
```

(continues on next page)

(continued from previous page)

```
                "city": "Jamietown"
            },
            "car": "mustang"
        },
        {
            "name": "Michelle Best",
            "age": 70,
            "address": {
                "city": "Port Dustin"
            },
            "car": ""
        },
        {
            "name": "Donald Hernandez",
            "age": 79,
            "address": {
                "city": "East Julieshire"
            },
            "car": "mustang"
        },
        {
            "name": "Kaitlyn Cook",
            "age": 3,
            "address": {
                "city": "Rachelton"
            },
            "car": "mustang"
        }
    ]
}
```

1.1.11 bool

Boolean value - *true* or *false*.

Arguments

None.

Example

```
{
    "awesome": "{{bool}}"
}

# Example output

{
    "awesome": "true"
}
```

1.1.12 option(arg1,arg2,...,argN)

Randomly choose one of the arguments (arg1,arg2,...,argN). This is very useful to generate more randomised data.

Arguments

Each argument can be an arbitrary value or even another placeholder.

Example

```
{
    "car": "{{option(mustang,{{empty}},C4)}}"
}

# Example output

{
    "car": "mustang"
}

# ... or for example

{
    "car": ""
}
```

Note: Missing some placeholder? JDOG can be easily *extended*.

CHAPTER

TWO

EXTENDING JDOG

It's easy to add new placeholder.

JDOG instance has two methods which support you to extend behavior.

- `add_matcher()` - adds new matcher and placeholder behavior
- `placeholder_keys()` - returns already defined placeholders

Each **placeholder** is represented by some subclass of `Placeholder` class.

Especially comes handy `FuncPlaceholder` which can be easily used to introduce new placeholders.

2.1 So, how to add a new placeholder?

For example we want to introduce *fizzbuzz* placeholder which with 50% chance print 'fizz' or 'buzz'.

Its only a few steps and you are good to go.

1. Think of a new name (or use an existing one).
2. Create regex pattern.
3. Add placeholder - use `FuncPlaceholder` or subclass `Placeholder`.
4. Put it together and call `add_matcher()`.

2.1.1 Come up with new name

It should be clear, only by name, what placeholder does. Beside the name think also about arguments.

> **Warning:** If you use existing name new behavior will replace odl one.

2.1.2 Create regex pattern

During parsing phase each placeholder is *tokenized*. Tokenization process use regex to match each placeholder.

Starting regex could look like **`^{{token}}$`**, that is:

- It has to start with double {{
- It has to end with double }}

If you want any arguments for the placeholder, regex **has to capture** arguments as a group, that is *^{{token((.*))$*.

A few examples of existing placeholders:

```
# age
r'^{{age}}$'

# number
r'^{{number\((.*)\)}}$'

# name (most complex one, arguments are optional)
r'^{{name\(?([f,m]?)\)?}}$'

# and new one
r'^{{fizzbuzz}}$'
```

Simple, isn't it? (If in doubt, take a look here.)

2.1.3 Add placeholder

Placeholder is special class that holds logic of generating specific values.

The easiest way is to use *FuncPlaceholder*.

- Takes argument - function.
- This function takes one argument - placeholder *arguments* as list.

So to our fizzbazz example:

```
def fizzbuzz(args):
    if random.random() > 0.5:
        return 'fizz'
    return 'buzz'
```

If you want more fine grained functionality, just subclass *Placeholder* and use it.

Note: If you want to automatically enclose returned value by placeholder within double quotes use *FuncStrPlaceholder*.

2.1.4 Putting it together

We have *name*, *regex* pattern and function which has logic of our *fizzbuzz placeholder*

On the instance of *Jdog* call *add_matcher()* function. Function takes three arguments

- **key** - the unique identification of placeholder - name.
- **pattern** - our regex pattern.
- **f_placeholder** - function which takes two arguments - token, it's arguments and should return *Placeholder* subclass.

Putting it together

```
# our pattern
pattern = r'^{{fizzbuzz}}$'

# placeholder logic
def fizzbuzz(args):
    if random.random() > 0.5:
        return 'fizz'
    return 'buzz'

# helper function to create placeholder
def create_fizzbuzz(token, args):
    return FuncStrPlaceholder(token, args, fizzbuzz)

jdog = Jdog()
jdog.add_matcher('fizzbuzz', pattern, create_fizzbuzz)
```

Warning: We are using *FuncStrPlaceholder* to automatically enclose value within double quotes. If you generate string values and do not enclose them result is not valid JSON.

Example can be simplified using lambda expressions.

```
jdog.add_matcher('fizzbuzz',match_fizzbuzz, lambda token, args:␣
↪FuncStrPlaceholder(token, args, fizzbuzz))
```

We can go further

```
# in fizzbuzz logic, we dont really care about arguments
jdog.add_matcher('fizzbuzz',match_fizzbuzz, lambda token, args:␣
↪FuncStrPlaceholder(token, args,lambda _: 'fizz' if random.random() > 0.5 else 'buzz
↪'))
```

But remember less lines does not mean more readable code. In this example rather opposite.

CHAPTER

THREE

API

3.1 JDOG

class jdog.jdog.**Jdog**(*lang='en-US'*, *strict=False*)

Proxy class to parser. Accepts language to use, parsing scheme and generating new data

Variables

- **lang** (*str*) – Language code to use. See `faker.Faker` for supported languages.
- **strict** (*boolean*) – If parser should raise NoMatchingPlaceholder error when parsing placeholders.

add_matcher(*key*, *pattern*, *f_placeholder*)

Add or redefine placeholder identified by KEY

Parameters

- **key** (*str*) – Unique placeholder identification.
- **pattern** (*str*) – Regex pattern which detects given placeholder.
- **f_placeholder** (*func*) – Function which should return `Placeholder` object. Function takes matched token and its arguments - if present.

generate()

Generate new data instance

Returns Generated JSON

Return type str

parse_scheme(*scheme*)

Parse scheme for generator

Parameters **scheme** (*str*) – Scheme to use and parse

Raises *jdog.parser.NoMatchingPlaceholder*, `json.JsonDecodeError`

placeholder_keys()

Returns all defined placeholder keys

Returns List of placeholder keys

Return type list

3.2 SchemeParser

exception jdog.parser.**NoMatchingPlaceholder**(*token*)
Raised when no matching placeholder was found.

class jdog.parser.**SchemeParser**(*lang='en-US'*, *strict=False*)
Parsing provided scheme and construct node structure for later data generation. Defines predefined placeholders.

Variables

- **faker** (*faker*) – Faker instance with provided language
- **strict** (*boolean*) – If parser should raise *NoMatchingPlaceholder*

add_matcher(*key*, *f_matcher*, *f_placeholder*)

Add new matcher for returning new placeholder

Parameters

- **key** – Unique matcher key. If provided existing one, old behavior is replaced.
- **f_matcher** – Function which takes one str argument. Should return re.Match object or None if no match found.
- **f_placeholder** – Function which takes accepted token and its parsed arguments - if present. Should return one of *Placeholder* object.

parse(*scheme*)

Parse given scheme and return node structure representation of the scheme.

Parameters **scheme** (*str*) – Scheme to parse

Returns Node structure representing current scheme

placeholder_keys()

Defined placeholder keys

Returns Defined placeholder keys

Return type list

3.3 Nodes

class jdog.node.**ArrayNode**(*children*)
Represents JSON-Array

Variables **properties** (*list*) – Values to include within object.

class jdog.node.**FuncNode**(*f*)
Node accepting function which will be executed later

Variables **f** (*func*) – Function to execute. Has no arguments.

exec()
Executes f

class jdog.node.**ObjectNode**(*properties*)
Represents JSON-Object

Variables **properties** (*list*) – Properties to include within object.

class jdog.node.**PlaceholderNode**(*placeholder*)
Represents node with object

Variables **placeholder** (*object*) – *Placeholder* object to use.

exec()

Executes placeholder

Returns Value returned by placeholder

class jdog.node.**PropertyNode**(*name*, *child*)
Represent JSON property with name and value

Variables

- **name** (*Node*) – Property name
- **child** (*Node*) – Property value

exec()
Executes name node and child if present. :return: Property representation

class jdog.node.**RangeNode**(*name*, *l*, *child*, *h=None*)
Node which repeatedly creates sub-nodes. Combination of property with array value.

Variables

- **name** (*str*) – Represents property name which will be generated.
- **l** (*int*) – How many times generation runs.
- **child** (*Node*) – Object to add to the generated array.
- **h** (*int*) – Optional. If specified, generation runs randomly between l (inlcusive) to h (exclusive).

exec()

Returns Range representation: JSON property named by `value` with array value.

Return type str

class jdog.node.**ScalarNode**(*value*)
Represents scalar value - that is any number or any arbitrary string

Variables **value** (*str|number*) – Scalar value

exec()
If value is string - return as JSON string value otherwise as is :return: Value enclosed as a string or as is

class jdog.node.**_GroupNode**(*begin_token*, *end_token*, *nodes*)
Base class for nodes which groups other nodes together.

Variables

- **begin_token** (*str*) – char to print at the start of group
- **end_token** (*str*) – char to print at the end of group
- **nodes** (*list*) – Nodes to recursively included within group

exec()
Join nodes together between `begin_token` and `end_token`

3.4 Placeholders

class jdog.placeholder.placeholder.**FuncPlaceholder**(*full_name*, *args*, *func*)
Represents placeholder which takes function to execute later. Returned value is fully determined by func itself.

Variables

- **func** (*func*) – Function to execute. Takes provided arguments from placeholder.
- **args** (*list*) – Parsed arguments as list

exec()
Executes func and return its returned value

class jdog.placeholder.placeholder.**FuncStrPlaceholder**(*full_name*, *args*, *func*)
Represents placeholder which takes function to execute later. The returned value is enclosed with double quotes to denote JSON string value.

Variables **func** (*func*) – Function to execute. Takes provided arguments from placeholder.

exec()
Executes func and return its value enclosed as string

class jdog.placeholder.placeholder.**Placeholder**(*full_name*, *arguments*)
Base class for placeholders.

Variables

- **full_name** (*str*) –
- **arguments** (*list*) – Arguments for placeholder

exec()
Each placeholder must have exec function

- JDOG is a Python library which helps generate a sample data for your projects.
- JDOG can also be run as CLI tool.
- For generating a sample data, the data scheme is provided.

CHAPTER

FOUR

SCHEME

- The *scheme* is provided in JSON format with special placeholders.
- In the output, the placeholders are replaced with some generated data.

Any valid JSON is **valid** scheme.

CHAPTER

FIVE

HOW TO USE IT?

Install it

```
python -m pip install jdog
```

Prepare a scheme

```
{
  "{{range(people,4)}}": {
    "name": "{{name}}",
    "age": "{{age}}",
    "address": {
      "city": "{{city}}"
    },
    "car": "{{option(mustang,{{empty}})}}"
  }
}
```

Use it

```
from jdog import Jdog

def main():
    jdog = Jdog()
    scheme = ... # your loaded scheme

    # parse scheme
    jdog.parse_scheme(scheme)

    # generate instance
    result = jdog.generate()

    print(result) # result is JSON
```

And the example result:

```
{
    "people": [
        {
            "name": "Brandi Young",
            "age": 39,
            "address": {
                "city": "Jamietown"
            },
            "car": "mustang"
```

(continues on next page)

(continued from previous page)

```
        },
        {
            "name": "Michelle Best",
            "age": 70,
            "address": {
                "city": "Port Dustin"
            },
            "car": ""
        },
        {
            "name": "Donald Hernandez",
            "age": 79,
            "address": {
                "city": "East Julieshire"
            },
            "car": "mustang"
        },
        {
            "name": "Kaitlyn Cook",
            "age": 3,
            "address": {
                "city": "Rachelton"
            },
            "car": "mustang"
        }
    ]
}
```

5.1 CLI

Package can be used as cli tool.

```
Usage: jdog [OPTIONS] SCHEME

Accepts SCHEME and generates new data to stdin or to specified OUTPUT

Options:
  -p, --pretty           Output as pretty JSON.
  -s, --strict           Raise error when no matching placeholder is found.
  -l, --lang TEXT        Language to use.
  --lang-help            Displays available language codes and exit.
  -o, --output FILENAME  Output file where result is written.
  --help                 Show this message and exit.
```

By default, CLI tool does not save output to file, just print results to standard output.

PYTHON MODULE INDEX

j

INDEX

Symbols

A

E

F

G

J

N

O

P

R

S