# Package 'zipangu'

February 7, 2021

**Title** Japanese Utility Functions and Data

**Version** 0.2.3

**Description** Some data treated by the Japanese R user require unique operations and processing. These are caused by address, Kanji, and traditional year representations. 'zipangu' transforms specific to Japan into something more general one.

**License** MIT + file LICENSE

**URL** https://uribo.github.io/zipangu/, https://github.com/uribo/zipangu

**BugReports** https://github.com/uribo/zipangu/issues

**Depends** R (¿= 3.2)

**Imports** dplyr (¿= 0.8.3), lifecycle (¿= 0.1.0), lubridate (¿= 1.7.4), magrittr (¿= 1.5), purrr (¿= 0.3.3), rlang (¿= 0.4.0), stringi (¿= 1.4.3), stringr (¿= 1.4.0), tibble (¿= 2.1.3), arabic2kansuji (¿= 0.1.0), stats

**Suggests** covr (¿= 3.4.0), testthat (¿= 2.1.0), scales (¿= 1.1.0)

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.1.1

## R topics documented:

---

`convert_jdate` *Convert Japanese date format to date object*

---

## Description

**Maturing**

## Usage

```
convert_jdate(date)
```

## Arguments

| | |
|---|---|
| `date` | a character object. |

## Examples

```
convert_jdate("\u4ee4\u548c2\u5e747\u67086\u65e5")

## [1] "2020-07-06"
```

---

`convert_jyear` *Convert Japanese imperial year to Anno Domini*

---

## Description

**Maturing**

## Usage

```
convert_jyear(jyear)
```

## Arguments

| | |
|---|---|
| `jyear` | Japanese imperial year (jyear). Kanji or Roman character |

### Examples

```
convert_jyear("R1")

## [1] 2019

convert_jyear("Heisei2")

## [1] 1990

convert_jyear("\u5e73\u6210\u5143\u5e74")

## [1] 1989

convert_jyear(c("\u662d\u548c10\u5e74", "\u5e73\u621014\u5e74"))

## [1] 1935 2002

convert_jyear(kansuji2arabic_all("\u5e73\u6210\u4e09\u5e74"))

## [1] 1991
```

---

## dl_zipcode_file *Download a zip-code file*

---

### Description

**Maturing**

### Usage

```
dl_zipcode_file(path, exdir = NULL)
```

### Arguments

| | |
|---|---|
| path | local file path or zip file URL |
| exdir | The directory to extract zip file. If NULL, use temporary folder. |

### Examples

```
## Not run:
dl_zipcode_file(path = "https://www.post.japanpost.jp/zipcode/dl/oogaki/zip/02aomori.zip")
dl_zipcode_file("https://www.post.japanpost.jp/zipcode/dl/oogaki/zip/02aomori.zip",
                exdir = getwd())

## End(Not run)
```

## find_date_by_wday *Find out the date of the specific month and weekday*

### Description

**Experimental** Get the date of the Xth the specific weekday

### Usage

```
find_date_by_wday(year, month, wday, ordinal)
```

### Arguments

| | |
|---|---|
| year | numeric year |
| month | numeric month |
| wday | numeric weekday |
| ordinal | number of week |

### Value

a vector of class POSIXct

### Examples

```
find_date_by_wday(2021, 1, 2, 2)
```

## is_jholiday *Is x a public holidays in Japan?*

### Description

**Experimental** Whether it is a holiday defined by Japanese law (enacted in 1948)

### Usage

```
is_jholiday(date)
```

### Arguments

| | |
|---|---|
| date | a vector of POSIXt, numeric or character objects |

### Details

Holiday information refers to data published as of December 21, 2020. Future holidays are subject to change.

### Value

TRUE if x is a public holidays in Japan, FALSE otherwise.

## Examples

```
is_jholiday("2021-01-01")
```

```
## [1] TRUE
```

```
is_jholiday("2018-12-23")
```

```
## [1] TRUE
```

```
is_jholiday("2019-12-23")
```

```
## [1] FALSE
```

---

`is_zipcode` *Test zip-code*

---

## Description

**Experimental**

## Usage

```
is_zipcode(x)
```

## Arguments

| | |
|---|---|
| `x` | Zip-code. Number or character. Hyphens may be included, but the input must contain a 7-character number. |

## Value

A logical vector.

## Examples

```
is_zipcode(7000027)
is_zipcode("700-0027")
```

jholiday_spec *Public holidays in Japan*

## Description

**Experimental**

## Usage

```
jholiday_spec(year, name, lang = "en")

jholiday(year, lang = "en")
```

## Arguments

| | |
|---|---|
| year | numeric year and in and after 1949. |
| name | holiday name |
| lang | return holiday names to "en" or "jp". |

## Details

Holiday information refers to data published as of December 21, 2020. Future holidays are subject to change.

## Examples

```
jholiday_spec(2019, "Sports Day")

## [1] "2019-10-14"

jholiday_spec(2021, "Sports Day")

## [1] "2021-07-23"
```

List of a specific year holidays

```
jholiday(2021, "en")

## $`New Year's Day`
## [1] "2021-01-01"
##
## $`Coming of Age Day`
## [1] "2021-01-11"
##
## $`Foundation Day`
## [1] "2021-02-11"
##
## $`The Emperor's Birthday`
## [1] "2021-02-23"
##
## $`Vernal Equinox Day`
```

```
## [1] "2021-03-20"
##
## $`Showa Day`
## [1] "2021-04-29"
##
## $`Constitution Memorial Day`
## [1] "2021-05-03"
##
## $`Greenery Day`
## [1] "2021-05-04"
##
## $`Children's Day`
## [1] "2021-05-05"
##
## $`Marine Day`
## [1] "2021-07-22"
##
## $`Sports Day`
## [1] "2021-07-23"
##
## $`Mountain Day`
## [1] "2021-08-08"
##
## $`Respect for the Aged Day`
## [1] "2021-09-20"
##
## $`Autumnal Equinox Day`
## [1] "2021-09-23"
##
## $`Culture Day`
## [1] "2021-11-03"
##
## $`Labour Thanksgiving Day`
## [1] "2021-11-23"
```

## References

Public Holiday Law https://www8.cao.go.jp/chosei/shukujitsu/gaiyou.html, https://elaws.e-gov.go.jp/document?lawid=323AC1000000178

---

`jpnprefs` *Prefectural informations in Japan*

---

## Description

Prefectures dataset.

## Usage

```
jpnprefs
```

### Format

A tibble with 47 rows 5 variables:

- jis_code: jis code
- prefecture_kanji: prefecture names
- prefecture: prefecture names
- region: region
- major_island:

### Examples

```
jpnprefs
```

---

`kansuji2arabic` *Convert kansuji character to arabic*

---

### Description

**Experimental** Converts a given Kansuji element such as Ichi (1) and Nana (7) to an Arabic. `kansuji2arabic_all()` converts only Kansuji in the string. `kansuji2arabic_num()` convert kansuji that contain the positions (e.g. Hyaku, Sen, etc) with the numbers represented by kansuji. `kansuji2arabic_str()` converts kansuji in a string to numbers represented by kansuji while retaining the non-kansuji characters.

### Usage

```
kansuji2arabic(str, convert = TRUE, .under = Inf)

kansuji2arabic_all(str, ...)

kansuji2arabic_num(str, consecutive = c("convert", "non"), ...)

kansuji2arabic_str(
  str,
  consecutive = c("convert", "non"),
  widths = c("all", "halfwidth"),
  ...
)
```

### Arguments

| | |
|---|---|
| `str` | Input vector. |
| `convert` | If `FALSE`, will return as numeric. The default value is `TRUE`, and numeric values are treated as strings. |
| `.under` | Number scale to be converted. The default value is infinity. |
| `...` | Other arguments to carry over to `kansuji2arabic()` |
| `consecutive` | If you select "convert", any sequence of 1 to 9 kansuji will be replaced with Arabic numerals. If you select "non", any sequence of 1-9 kansuji will not be replaced by Arabic numerals. |

widths — If you select "all", both full-width and half-width Arabic numerals are taken into account when calculating kansuji, but if you select "halfwidth", only half-width Arabic numerals are taken into account when calculating kansuji.

## Value

a character or numeric.

## Examples

```
kansuji2arabic("\u4e00")
kansuji2arabic(c("\u4e00", "\u767e"))
kansuji2arabic(c("\u4e00", "\u767e"), convert = FALSE)
# Keep Kansuji over 1000.
kansuji2arabic(c("\u4e00", "\u767e", "\u5343"), .under = 1000)
# Convert all character
kansuji2arabic_all("\u3007\u4e00\u4e8c\u4e09\u56db\u4e94\u516d\u4e03\u516b\u4e5d\u5341")
kansuji2arabic_all("\u516b\u4e01\u76ee")
# Convert kansuji that contain the positions with the numbers represented by kansuji.
kansuji2arabic_num("\u4e00\u5104\u4e8c\u5343\u4e09\u767e\u56db\u5341\u4e94\u4e07")
kansuji2arabic_num("\u4e00\u5104\u4e8c\u4e09\u56db\u4e94\u4e07\u516d\u4e03\u516b\u4e5d")
# Converts kansuji in a string to numbers represented by kansuji.
kansuji2arabic_str("\u91d1\u4e00\u5104\u4e8c\u5343\u4e09\u767e\u56db\u5341\u4e94\u4e07\u5186")
kansuji2arabic_str("\u91d1\u4e00\u5104\u4e8c\u4e09\u56db\u4e94\u4e07\u516d\u4e03\u516b\u4e5d\u5186")
kansuji2arabic_str("\u91d11\u51042345\u4e076789\u5186")
```

---

# label_kansuji *Label numbers in Kansuji format*

---

## Description

Automatically scales and labels with the Kansuji Myriad Scale (e.g. "Man", "Oku", etc). Use label_kansuji() converts the label value to either Kansuji value or a mixture of Arabic numerals and the Kansuji Scales for ten thousands, billions, and ten quadrillions. Use label_kansuji_suffix() converts the label value to an Arabic numeral followed by the Kansuji Scale with the suffix.

## Usage

```
label_kansuji(
  unit = NULL,
  sep = "",
  prefix = "",
  big.mark = "",
  number = c("arabic", "kansuji"),
  ...
)

label_kansuji_suffix(
  accuracy = 1,
  unit = NULL,
  sep = NULL,
```

```
  prefix = "",
  big.mark = "",
  significant.digits = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| unit | Optional units specifier. |
| sep | Separator between number and Kansuji unit. |
| prefix | Symbols to display before value. |
| big.mark | Character used between every 3 digits to separate thousands. |
| number | If Number is arabic, it will return a mixture of Arabic and the Kansuji Myriad Scale; if Kansuji, it will return only Kansuji numerals. |
| ... | Other arguments passed on to base::prettyNum() or scales::label_number(). |
| accuracy | A number to round to. Use (e.g.) 0.01 to show 2 decimal places of precision. |
| significant.digits | Determines whether or not the value of accurary is valid as a significant figure with a decimal point. The default is FALSE, in which case if accurary is 2 and the value is 1.10, 1.1 will be displayed, but if TRUE and installed 'scales' package, 1.10 will be displayed. |

### Value

All label_() functions return a "labelling" function, i.e. a function that takes a vector x and returns a character vector of length(x) giving a label for each input value.

### Examples

```
library("scales")
demo_continuous(c(1, 1e9), label = label_kansuji())
demo_continuous(c(1, 1e9), label = label_kansuji_suffix())
```

---

## read_zipcode *Read Japan post's zip-code file*

---

### Description

**Experimental**

### Usage

```
read_zipcode(path, type = c("oogaki", "kogaki", "roman", "jigyosyo"))
```

### Arguments

| | |
|---|---|
| path | local file path or zip file URL |
| type | Input file type, one of "oogaki", "kogaki", "roman", "jigyosyo" |

### Details

Reads zip-code data in csv format provided by japan post group and parse it as a data.frame. Corresponds to the available "oogaki", "kogaki", "roman" and "jigyosyo" types. These file types must be specified by the argument.

### Value

tibble

### See Also

https://www.post.japanpost.jp/zipcode/dl/readme.html, https://www.post.japanpost.jp/zipcode/dl/jigyosyo/readme.html

### Examples

```
# Input sources
read_zipcode(path = system.file("zipcode_dummy/13TOKYO_oogaki.CSV", package = "zipangu"),
             type = "oogaki")
read_zipcode(system.file("zipcode_dummy/13TOKYO_kogaki.CSV", package = "zipangu"),
             "oogaki")
read_zipcode(system.file("zipcode_dummy/KEN_ALL_ROME.CSV", package = "zipangu"),
             "roman")
read_zipcode(system.file("zipcode_dummy/JIGYOSYO.CSV", package = "zipangu"),
             "jigyosyo")
## Not run:
# Or directly from a URL
read_zipcode("https://www.post.japanpost.jp/zipcode/dl/jigyosyo/zip/jigyosyo.zip")

## End(Not run)
```

---

## separate_address *Separate address elements*

---

### Description

**Experimental** Parses and decomposes address string into elements of prefecture, city, and lower address.

### Usage

```
separate_address(str)
```

### Arguments

| | |
|---|---|
| str | Input vector. address strings. |

### Value

A list of elements that make up an address.

### Examples

```
separate_address("\u5317\u6d77\u9053\u672d\u5e4c\u5e02\u4e2d\u592e\u533a")
```

str_jconv *Converts the kind of string used as Japanese*

## Description

**Stable**

## Usage

```
str_jconv(str, fun, to)

str_conv_hirakana(str, to = c("hiragana", "katakana"))

str_conv_zenhan(str, to = c("zenkaku", "hankaku"))

str_conv_romanhira(str, to = c("roman", "hiragana"))

str_conv_normalize(str, to = c("nfkc"))
```

## Arguments

| | |
|---|---|
| str | Input vector. |
| fun | convert function |
| to | Select the type of character to convert. |

## Details

Converts the types of string treat by Japanese people to each other. The following types are supported.

- Hiraganra to Katakana
- Zenkaku to Hankaku
- Latin (Roman) to Hiragana

## See Also

These functions are powered by the stringi package's stri_trans_general().

## Examples

```
str_jconv("\u30a2\u30a4\u30a6\u30a8\u30aa", str_conv_hirakana, to = "hiragana")
str_jconv("\u3042\u3044\u3046\u3048\u304a", str_conv_hirakana, to = "katakana")
str_jconv("\uff41\uff10", str_conv_zenhan, "hankaku")
str_jconv("\uff76\uff9e\uff6f", str_conv_zenhan, "zenkaku")
str_jconv("\u30a2\u30a4\u30a6\u30a8\u30aa", str_conv_romanhira, "roman")
str_jconv("\u2460", str_conv_normalize, "nfkc")
str_conv_hirakana("\u30a2\u30a4\u30a6\u30a8\u30aa", to = "hiragana")
str_conv_hirakana("\u3042\u3044\u3046\u3048\u304a", to = "katakana")
str_conv_zenhan("\uff41\uff10", "hankaku")
str_conv_zenhan("\uff76\uff9e\uff6f", "zenkaku")
str_conv_romanhira("aiueo", "hiragana")
str_conv_romanhira("\u3042\u3044\u3046\u3048\u304a", "roman")
str_conv_normalize("\u2460", "nfkc")
```

`zipcode_spacer` *Insert and remove zip-code connect character*

## Description

**Maturing** Inserts a hyphen as a delimiter in the given zip-code string. Or exclude the hyphen.

## Usage

```
zipcode_spacer(x, remove = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | Zip-code. Number or character. Hyphens may be included, but the input must contain a 7-character number. |
| `remove` | Default is `FALSE`. If `TRUE`, remove the hyphen. |

## Examples

```
zipcode_spacer(7000027)
zipcode_spacer("305-0053")
zipcode_spacer("305-0053", remove = TRUE)
```

# Index