Inhaltsverzeichnis

Modul:Unicode data/combining

Contains data used by Module:Unicode data to determine whether a character is combining or not. Can also be used to return the character's combining class. Generated from DerivedCombiningClass.txt using a script.

```
local data = {
        singles = {
                [0x000315] = 232,
                [0x00031A] = 232,
                [0x00031B] = 216,
                [0x000345] = 240,
                [0x000346] = 230,
                [0x000357] = 230,
                [0x000358] = 232,
                [0x00035B] = 230,
                [0x00035C] = 233,
                [0x00035F] = 233,
                [0x000362] = 233,
                [0x000591] = 220,
                [0x000596] = 220,
                [0x00059A] = 222,
                [0x00059B] = 220,
                [0x0005AA] = 220,
                [0x0005AD] = 222,
                [0x0005AE] = 228,
                [0x0005AF] = 230,
                [0x0005B0] =  10,
                [0x0005B1] =  11,
                [0x0005B2] =  12,
                [0x0005B3] =  13,
                [0x0005B4] =  14,
                [0x0005B5] =  15,
                [0x0005B6] =  16,
                [0x0005B7] =  17,
                [0x0005B8] =  18,
                [0x0005BB] =  20,
                [0x0005BC] =  21,
                [0x0005BD] =  22,
                [0x0005BF] =  23,
                [0x0005C1] =  24,
                [0x0005C2] =  25,
                [0x0005C4] = 230,
                [0x0005C5] = 220,
                [0x0005C7] =  18,
                [0x000618] =  30,
                [0x000619] =  31,
                [0x00061A] =  32,
                [0x00064B] =  27,
                [0x00064C] =  28,
                [0x00064D] =  29,
                [0x00064E] =  30,
                [0x00064F] =  31,
                [0x000650] =  32,
                [0x000651] =  33,
                [0x000652] =  34,
                [0x00065C] = 220,
                [0x00065F] = 220,
                [0x000670] =  35,
```

```
                [0x0006E3] = 220,
                [0x0006E4] = 230,
                [0x0006EA] = 220,
                [0x0006ED] = 220,
                [0x000711] =  36,
                [0x000730] = 230,
                [0x000731] = 220,
                [0x000734] = 220,
                [0x00073A] = 230,
                [0x00073D] = 230,
                [0x00073E] = 220,
                [0x000742] = 220,
                [0x000743] = 230,
                [0x000744] = 220,
                [0x000745] = 230,
                [0x000746] = 220,
                [0x000747] = 230,
                [0x000748] = 220,
                [0x0007F2] = 220,
                [0x0007F3] = 230,
                [0x0007FD] = 220,
                [0x000898] = 230,
                [0x0008E3] = 220,
                [0x0008E6] = 220,
                [0x0008E9] = 220,
                [0x0008F0] =  27,
                [0x0008F1] =  28,
                [0x0008F2] =  29,
                [0x0008F6] = 220,
                [0x00093C] =   7,
                [0x00094D] =   9,
                [0x000951] = 230,
                [0x000952] = 220,
                [0x0009BC] =   7,
                [0x0009CD] =   9,
                [0x0009FE] = 230,
                [0x000A3C] =   7,
                [0x000A4D] =   9,
                [0x000ABC] =   7,
                [0x000ACD] =   9,
                [0x000B3C] =   7,
                [0x000B4D] =   9,
                [0x000BCD] =   9,
                [0x000C3C] =   7,
                [0x000C4D] =   9,
                [0x000C55] =  84,
                [0x000C56] =  91,
                [0x000CBC] =   7,
                [0x000CCD] =   9,
                [0x000D4D] =   9,
                [0x000DCA] =   9,
                [0x000E3A] =   9,
                [0x000EBA] =   9,
                [0x000F35] = 220,
                [0x000F37] = 220,
                [0x000F39] = 216,
                [0x000F71] = 129,
                [0x000F72] = 130,
                [0x000F74] = 132,
                [0x000F80] = 130,
                [0x000F84] =   9,
                [0x000FC6] = 220,
                [0x001037] =   7,
                [0x00108D] = 220,
```

```
                [0x001714] =     9,
                [0x001715] =     9,
                [0x001734] =     9,
                [0x0017D2] =     9,
                [0x0017DD] =   230,
                [0x0018A9] =   228,
                [0x001939] =   222,
                [0x00193A] =   230,
                [0x00193B] =   220,
                [0x001A17] =   230,
                [0x001A18] =   220,
                [0x001A60] =     9,
                [0x001A7F] =   220,
                [0x001ABD] =   220,
                [0x001ACA] =   220,
                [0x001B34] =     7,
                [0x001B44] =     9,
                [0x001B6B] =   230,
                [0x001B6C] =   220,
                [0x001BAA] =     9,
                [0x001BAB] =     9,
                [0x001BE6] =     7,
                [0x001C37] =     7,
                [0x001CD4] =     1,
                [0x001CE0] =   230,
                [0x001CED] =   220,
                [0x001CF4] =   230,
                [0x001DC2] =   220,
                [0x001DCA] =   220,
                [0x001DCD] =   234,
                [0x001DCE] =   214,
                [0x001DCF] =   220,
                [0x001DD0] =   202,
                [0x001DF6] =   232,
                [0x001DF9] =   220,
                [0x001DFA] =   218,
                [0x001DFB] =   230,
                [0x001DFC] =   233,
                [0x001DFD] =   220,
                [0x001DFE] =   230,
                [0x001DFF] =   220,
                [0x0020E1] =   230,
                [0x0020E7] =   230,
                [0x0020E8] =   220,
                [0x0020E9] =   230,
                [0x0020F0] =   230,
                [0x002D7F] =     9,
                [0x00302A] =   218,
                [0x00302B] =   228,
                [0x00302C] =   232,
                [0x00302D] =   222,
                [0x00A66F] =   230,
                [0x00A806] =     9,
                [0x00A82C] =     9,
                [0x00A8C4] =     9,
                [0x00A953] =     9,
                [0x00A9B3] =     7,
                [0x00A9C0] =     9,
                [0x00AAB0] =   230,
                [0x00AAB4] =   220,
                [0x00AAC1] =   230,
                [0x00AAF6] =     9,
                [0x00ABED] =     9,
                [0x00FB1E] =    26,
```

```
                    [0x0101FD] = 220,
                    [0x0102E0] = 220,
                    [0x010A0D] = 220,
                    [0x010A0F] = 230,
                    [0x010A38] = 230,
                    [0x010A39] =   1,
                    [0x010A3A] = 220,
                    [0x010A3F] =   9,
                    [0x010AE5] = 230,
                    [0x010AE6] = 220,
                    [0x010F4B] = 220,
                    [0x010F4C] = 230,
                    [0x010F82] = 230,
                    [0x010F83] = 220,
                    [0x010F84] = 230,
                    [0x010F85] = 220,
                    [0x011046] =   9,
                    [0x011070] =   9,
                    [0x01107F] =   9,
                    [0x0110B9] =   9,
                    [0x0110BA] =   7,
                    [0x011173] =   7,
                    [0x0111C0] =   9,
                    [0x0111CA] =   7,
                    [0x011235] =   9,
                    [0x011236] =   7,
                    [0x0112E9] =   7,
                    [0x0112EA] =   9,
                    [0x01134D] =   9,
                    [0x011442] =   9,
                    [0x011446] =   7,
                    [0x01145E] = 230,
                    [0x0114C2] =   9,
                    [0x0114C3] =   7,
                    [0x0115BF] =   9,
                    [0x0115C0] =   7,
                    [0x01163F] =   9,
                    [0x0116B6] =   9,
                    [0x0116B7] =   7,
                    [0x01172B] =   9,
                    [0x011839] =   9,
                    [0x01183A] =   7,
                    [0x01193D] =   9,
                    [0x01193E] =   9,
                    [0x011943] =   7,
                    [0x0119E0] =   9,
                    [0x011A34] =   9,
                    [0x011A47] =   9,
                    [0x011A99] =   9,
                    [0x011C3F] =   9,
                    [0x011D42] =   7,
                    [0x011D97] =   9,
                    [0x01BC9E] =   1,
                    [0x01D16D] = 226,
                    [0x01E2AE] = 230,
                    [0x01E94A] =   7,
            };
            ranges = {
                    { 0x000300, 0x000314, 230 },
                    { 0x000316, 0x000319, 220 },
                    { 0x00031C, 0x000320, 220 },
                    { 0x000321, 0x000322, 202 },
                    { 0x000323, 0x000326, 220 },
                    { 0x000327, 0x000328, 202 },
```

```
                { 0x000329, 0x000333, 220 },
                { 0x000334, 0x000338,   1 },
                { 0x000339, 0x00033C, 220 },
                { 0x00033D, 0x000344, 230 },
                { 0x000347, 0x000349, 220 },
                { 0x00034A, 0x00034C, 230 },
                { 0x00034D, 0x00034E, 220 },
                { 0x000350, 0x000352, 230 },
                { 0x000353, 0x000356, 220 },
                { 0x000359, 0x00035A, 220 },
                { 0x00035D, 0x00035E, 234 },
                { 0x000360, 0x000361, 234 },
                { 0x000363, 0x00036F, 230 },
                { 0x000483, 0x000487, 230 },
                { 0x000592, 0x000595, 230 },
                { 0x000597, 0x000599, 230 },
                { 0x00059C, 0x0005A1, 230 },
                { 0x0005A2, 0x0005A7, 220 },
                { 0x0005A8, 0x0005A9, 230 },
                { 0x0005AB, 0x0005AC, 230 },
                { 0x0005B9, 0x0005BA,  19 },
                { 0x000610, 0x000617, 230 },
                { 0x000653, 0x000654, 230 },
                { 0x000655, 0x000656, 220 },
                { 0x000657, 0x00065B, 230 },
                { 0x00065D, 0x00065E, 230 },
                { 0x0006D6, 0x0006DC, 230 },
                { 0x0006DF, 0x0006E2, 230 },
                { 0x0006E7, 0x0006E8, 230 },
                { 0x0006EB, 0x0006EC, 230 },
                { 0x000732, 0x000733, 230 },
                { 0x000735, 0x000736, 230 },
                { 0x000737, 0x000739, 220 },
                { 0x00073B, 0x00073C, 220 },
                { 0x00073F, 0x000741, 230 },
                { 0x000749, 0x00074A, 230 },
                { 0x0007EB, 0x0007F1, 230 },
                { 0x000816, 0x000819, 230 },
                { 0x00081B, 0x000823, 230 },
                { 0x000825, 0x000827, 230 },
                { 0x000829, 0x00082D, 230 },
                { 0x000859, 0x00085B, 220 },
                { 0x000899, 0x00089B, 220 },
                { 0x00089C, 0x00089F, 230 },
                { 0x0008CA, 0x0008CE, 230 },
                { 0x0008CF, 0x0008D3, 220 },
                { 0x0008D4, 0x0008E1, 230 },
                { 0x0008E4, 0x0008E5, 230 },
                { 0x0008E7, 0x0008E8, 230 },
                { 0x0008EA, 0x0008EC, 230 },
                { 0x0008ED, 0x0008EF, 220 },
                { 0x0008F3, 0x0008F5, 230 },
                { 0x0008F7, 0x0008F8, 230 },
                { 0x0008F9, 0x0008FA, 220 },
                { 0x0008FB, 0x0008FF, 230 },
                { 0x000953, 0x000954, 230 },
                { 0x000D3B, 0x000D3C,   9 },
                { 0x000E38, 0x000E39, 103 },
                { 0x000E48, 0x000E4B, 107 },
                { 0x000EB8, 0x000EB9, 118 },
                { 0x000EC8, 0x000ECB, 122 },
                { 0x000F18, 0x000F19, 220 },
                { 0x000F7A, 0x000F7D, 130 },
                { 0x000F82, 0x000F83, 230 },
```

```
                { 0x000F86, 0x000F87, 230 },
                { 0x001039, 0x00103A,   9 },
                { 0x00135D, 0x00135F, 230 },
                { 0x001A75, 0x001A7C, 230 },
                { 0x001AB0, 0x001AB4, 230 },
                { 0x001AB5, 0x001ABA, 220 },
                { 0x001ABB, 0x001ABC, 230 },
                { 0x001ABF, 0x001AC0, 220 },
                { 0x001AC1, 0x001AC2, 230 },
                { 0x001AC3, 0x001AC4, 220 },
                { 0x001AC5, 0x001AC9, 230 },
                { 0x001ACB, 0x001ACE, 230 },
                { 0x001B6D, 0x001B73, 230 },
                { 0x001BF2, 0x001BF3,   9 },
                { 0x001CD0, 0x001CD2, 230 },
                { 0x001CD5, 0x001CD9, 220 },
                { 0x001CDA, 0x001CDB, 230 },
                { 0x001CDC, 0x001CDF, 220 },
                { 0x001CE2, 0x001CE8,   1 },
                { 0x001CF8, 0x001CF9, 230 },
                { 0x001DC0, 0x001DC1, 230 },
                { 0x001DC3, 0x001DC9, 230 },
                { 0x001DCB, 0x001DCC, 230 },
                { 0x001DD1, 0x001DF5, 230 },
                { 0x001DF7, 0x001DF8, 228 },
                { 0x0020D0, 0x0020D1, 230 },
                { 0x0020D2, 0x0020D3,   1 },
                { 0x0020D4, 0x0020D7, 230 },
                { 0x0020D8, 0x0020DA,   1 },
                { 0x0020DB, 0x0020DC, 230 },
                { 0x0020E5, 0x0020E6,   1 },
                { 0x0020EA, 0x0020EB,   1 },
                { 0x0020EC, 0x0020EF, 220 },
                { 0x002CEF, 0x002CF1, 230 },
                { 0x002DE0, 0x002DFF, 230 },
                { 0x00302E, 0x00302F, 224 },
                { 0x003099, 0x00309A,   8 },
                { 0x00A674, 0x00A67D, 230 },
                { 0x00A69E, 0x00A69F, 230 },
                { 0x00A6F0, 0x00A6F1, 230 },
                { 0x00A8E0, 0x00A8F1, 230 },
                { 0x00A92B, 0x00A92D, 220 },
                { 0x00AAB2, 0x00AAB3, 230 },
                { 0x00AAB7, 0x00AAB8, 230 },
                { 0x00AABE, 0x00AABF, 230 },
                { 0x00FE20, 0x00FE26, 230 },
                { 0x00FE27, 0x00FE2D, 220 },
                { 0x00FE2E, 0x00FE2F, 230 },
                { 0x010376, 0x01037A, 230 },
                { 0x010D24, 0x010D27, 230 },
                { 0x010EAB, 0x010EAC, 230 },
                { 0x010F46, 0x010F47, 220 },
                { 0x010F48, 0x010F4A, 230 },
                { 0x010F4D, 0x010F50, 220 },
                { 0x011100, 0x011102, 230 },
                { 0x011133, 0x011134,   9 },
                { 0x01133B, 0x01133C,   7 },
                { 0x011366, 0x01136C, 230 },
                { 0x011370, 0x011374, 230 },
                { 0x011D44, 0x011D45,   9 },
                { 0x016AF0, 0x016AF4,   1 },
                { 0x016B30, 0x016B36, 230 },
                { 0x016FF0, 0x016FF1,   6 },
                { 0x01D165, 0x01D166, 216 },
```

```
                { 0x01D167, 0x01D169,   1 },
                { 0x01D16E, 0x01D172, 216 },
                { 0x01D17B, 0x01D182, 220 },
                { 0x01D185, 0x01D189, 230 },
                { 0x01D18A, 0x01D18B, 220 },
                { 0x01D1AA, 0x01D1AD, 230 },
                { 0x01D242, 0x01D244, 230 },
                { 0x01E000, 0x01E006, 230 },
                { 0x01E008, 0x01E018, 230 },
                { 0x01E01B, 0x01E021, 230 },
                { 0x01E023, 0x01E024, 230 },
                { 0x01E026, 0x01E02A, 230 },
                { 0x01E130, 0x01E136, 230 },
                { 0x01E2EC, 0x01E2EF, 230 },
                { 0x01E8D0, 0x01E8D6, 220 },
                { 0x01E944, 0x01E949, 230 },
        };
}

-- Needed for binary search.
data.ranges.length = #data.ranges

return data
```

Modul:Unicode data

Usage

This module provides functions that access information on Unicode code points. The information is retrieved from data modules generated from the Unicode Character Database, or derived by rules given in the Unicode Specification. It and its submodules were copied from English Wiktionary and then modified; see there for more information.

Functions

Vorlage:Code

Receives a code point (number) and returns its name or label; for example, Vorlage:Code returns Vorlage:Code.

For example, Vorlage:Tnull → <reserved-0061>

Vorlage:Code

Template-invokable functions that allow access to the functions starting with `lookup` and `is`. Replace the first underscore in the function name with a pipe. For most of the functions, add the code point in hexadecimal base as the next parameter, but for `is_Latin`, `is_rtl`, and `is_valid_pagename`, add text. HTML character references in the text are decoded by the module into code points.

For example, Vorlage:Tnull → **Lua-Fehler in Zeile 293: attempt to index local 'data_module' (a boolean value)**.

Data modules

The data used by functions in this module is found in submodules. Some are generated by AWK scripts shown at User:Kephir/Unicode on English Wiktionary, others by Lua scripts on the /make subpages of the submodules.

- Module:Unicode data/aliases: the formal name aliases for characters (from NameAliases.txt)
- Module:Unicode data/blocks: the list of Unicode blocks (from Blocks.txt)
- Module:Unicode data/category: data mapping characters to their General Category (from DerivedGeneralCategory.txt)
- Module:Unicode data/control: data for identifying characters that belong to the General Categories of Separator and Other (from DerivedGeneralCategory.txt)

- Module:Unicode data/combining: data mapping characters to their Combining Classes (from DerivedCombiningClass.txt)
- Module:Unicode data/Hangul: data used to generate the names of Hangul syllables (from Jamo.txt)
- Module:Unicode data/scripts: data mapping characters to their Unicode script properties (from Scripts.txt).

The name data modules (Module:Unicode data/names/*xxx*) were compiled from UnicodeData.txt. Each one contains, at maximum, code points U+xxx000 to U+xxxFFF. **Lua-Fehler in mw.title.lua, Zeile 206: too many expensive function calls**

Copyright

The Unicode database is released by Unicode Inc. under the following terms:

```
local p = {}

local floor = math.floor

local function errorf(level, ...)
        if type(level) == "number" then
                return error(string.format(...), level + 1)
```

```lua
			else -- level is actually the format string.
				return error(string.format(level, ...), 2)
		end
	end

	local function binary_range_search(codepoint, ranges)
		local low, mid, high
		low, high = 1, ranges.length or require "Module:TableTools".length(ranges)
		while low <= high do
			mid = floor((low + high) / 2)
			local range = ranges[mid]
			if codepoint < range[1] then
				high = mid - 1
			elseif codepoint <= range[2] then
				return range, mid
			else
				low = mid + 1
			end
		end
		return nil, mid
	end
	p.binary_range_search = binary_range_search

	--[[
	local function linear_range_search(codepoint, ranges)
		for i, range in ipairs(ranges) do
			if range[1] <= codepoint and codepoint <= range[2] then
				return range
			end
		end
	end
	--]]

	-- Load a module by indexing "loader" with the name of the module minus the
	-- "Module:Unicode data/" part. For instance, loader.blocks returns
	-- [[Module:Unicode data/blocks]]. If a module cannot be loaded, false will be
	-- returned.
	local loader = setmetatable({}, {
		__index = function (self, key)
			local success, data = pcall(mw.loadData, "Module:Unicode data/"
			if not success then
				data = false
			end
			self[key] = data
			return data
		end
	})

	-- For the algorithm used to generate Hangul Syllable names,
	-- see "Hangul Syllable Name Generation" in section 3.12 of the
	-- Unicode Specification:
	-- https://www.unicode.org/versions/Unicode11.0.0/ch03.pdf
	local name_hooks = {
		{	  0x00,	  0x1F, "<control-%04X>" }, -- C0 control characters
		{	  0x7F,	  0x9F, "<control-%04X>" }, -- DEL and C1 control character
		{	0x3400,	0x4DBF, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK Ideograph Ex
		{	0x4E00,	0x9FFF, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK Ideograph
		{	0xAC00,	0xD7A3, function (codepoint) -- Hangul Syllables
			local Hangul_data = loader.Hangul
			local syllable_index = codepoint - 0xAC00

			return ("HANGUL SYLLABLE %s%s%s"):format(
				Hangul_data.leads[floor(syllable_index / Hangul_data.fina
				Hangul_data.vowels[floor((syllable_index % Hangul_data.fi
```

```lua
                                / Hangul_data.trail_count)],
                        Hangul_data.trails[syllable_index % Hangul_data.trail_cou
                )
        end },
        -- High Surrogates, High Private Use Surrogates, Low Surrogates
        {   0xD800,   0xDFFF, "<surrogate-%04X>" },
        {   0xE000,   0xF8FF, "<private-use-%04X>" }, -- Private Use
        -- CJK Compatibility Ideographs
        {   0xF900,   0xFA6D, "CJK COMPATIBILITY IDEOGRAPH-%04X" },
        {   0xFA70,   0xFAD9, "CJK COMPATIBILITY IDEOGRAPH-%04X" },
        {  0x17000,  0x187F7, "TANGUT IDEOGRAPH-%04X" }, -- Tangut Ideograph
        {  0x18800,  0x18AFF, function (codepoint)
                return ("TANGUT COMPONENT-%03d"):format(codepoint - 0x187FF)
        end },
        {  0x18D00,  0x18D08, "TANGUT IDEOGRAPH-%04X" }, -- Tangut Ideograph Supp
        {  0x1B170,  0x1B2FB, "NUSHU CHARACTER-%04X" }, -- Nushu
        {  0x20000,  0x2A6DF, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK Ideograph Ex
        {  0x2A700,  0x2B738, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK Ideograph Ex
        {  0x2B740,  0x2B81D, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK Ideograph Ex
        {  0x2B820,  0x2CEA1, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK Ideograph Ex
        {  0x2CEB0,  0x2EBE0, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK Ideograph Ex
        -- CJK Compatibility Ideographs Supplement (Supplementary Ideographic Pla
        {  0x2F800,  0x2FA1D, "CJK COMPATIBILITY IDEOGRAPH-%04X" },
        {  0xE0100,  0xE01EF, function (codepoint) -- Variation Selectors Supplem
                return ("VARIATION SELECTOR-%d"):format(codepoint - 0xE0100 + 17)
        end},
        {  0x30000,  0x3134A, "CJK UNIFIED IDEOGRAPH-%04X" }, -- CJK Ideograph Ex
        {  0xF0000,  0xFFFFD, "<private-use-%04X>" }, -- Plane 15 Private Use
        { 0x100000, 0x10FFFD, "<private-use-%04X>" }  -- Plane 16 Private Use
}
name_hooks.length = #name_hooks

local name_range_cache

local function generate_name(data, codepoint)
        if type(data) == "string" then
                return data:format(codepoint)
        else
                return data(codepoint)
        end
end


--[[
-- Checks that the code point is a number and in range.
-- Does not check whether code point is an integer.
-- Not used
local function check_codepoint(funcName, argIdx, val)
        require 'libraryUtil'.checkType(funcName, argIdx, val, 'number')
        if codepoint < 0 or 0x10FFFF < codepoint then
                errorf("Codepoint %04X out of range", codepoint)
        end
end
--]]

-- https://www.unicode.org/versions/Unicode11.0.0/ch04.pdf, section 4.8
function p.lookup_name(codepoint)
        -- U+FDD0-U+FDEF and all code points ending in FFFE or FFFF are Unassigne
        -- (Cn) and specifically noncharacters:
        -- https://www.unicode.org/faq/private_use.html#nonchar4
        if 0xFDD0 <= codepoint and (codepoint <= 0xFDEF
                        or floor(codepoint % 0x10000) >= 0xFFFE) then
                return ("<noncharacter-%04X>"):format(codepoint)
        end
```

```
                if name_range_cache -- Check if previously used "name hook" applies to th
                        and codepoint >= name_range_cache[1]
                        and codepoint <= name_range_cache[2] then
                return generate_name(name_range_cache[3], codepoint)
        end

        local range = binary_range_search(codepoint, name_hooks)
        if range then
                name_range_cache = range
                return generate_name(range[3], codepoint)
        end

        local data = loader[('names/%03X'):format(codepoint / 0x1000)]

        if data and data[codepoint] then
                return data[codepoint]

        -- Unassigned (Cn) consists of noncharacters and reserved characters.
        -- The character has been established not to be a noncharacter,
        -- and if it were assigned, its name would already been retrieved,
        -- so it must be reserved.
        else
                return ("<reserved-%04X>"):format(codepoint)
        end
end

--[[
-- No image data modules on Wikipedia yet.
function p.lookup_image(codepoint)
        local data = loader[('images/%03X'):format(codepoint / 0x1000)]

        if data then
                return data[codepoint]
        end
end
--]]

local planes = {
        [ 0] = "Basic Multilingual Plane";
        [ 1] = "Supplementary Multilingual Plane";
        [ 2] = "Supplementary Ideographic Plane";
        [ 3] = "Tertiary Ideographic Plane";
        [14] = "Supplementary Special-purpose Plane";
        [15] = "Supplementary Private Use Area-A";
        [16] = "Supplementary Private Use Area-B";
}

-- Load [[Module:Unicode data/blocks]] if needed and assign it to this variable.
local blocks

local function block_iter(blocks, i)
        i = i + 1
        local data = blocks[i]
        if data then
                -- Unpack doesn't work on tables loaded with mw.loadData.
                return i, data[1], data[2], data[3]
        end
end

-- An ipairs-type iterator generator for the list of blocks.
function p.enum_blocks()
        local blocks = loader.blocks
        return block_iter, blocks, 0
end
```

```lua
function p.lookup_plane(codepoint)
        local i = floor(codepoint / 0x10000)
        return planes[i] or ("Plane %u"):format(i)
end

function p.lookup_block(codepoint)
        local blocks = loader.blocks
        local range = binary_range_search(codepoint, blocks)
        if range then
                return range[3]
        else
                return "No Block"
        end
end

function p.get_block_info(name)
        for i, block in ipairs(loader.blocks) do
                if block[3] == name then
                        return block
                end
        end
end

function p.is_valid_pagename(pagename)
        local has_nonws = false

        for cp in mw.ustring.gcodepoint(pagename) do
                if (cp == 0x0023) -- #
                or (cp == 0x005B) -- [
                or (cp == 0x005D) -- ]
                or (cp == 0x007B) -- {
                or (cp == 0x007C) -- |
                or (cp == 0x007D) -- }
                or (cp == 0x180E) -- MONGOLIAN VOWEL SEPARATOR
                or ((cp >= 0x2000) and (cp <= 0x200A)) -- spaces in General Punct
                or (cp == 0xFFFD) -- REPLACEMENT CHARACTER
                then
                        return false
                end

                local printable, result = p.is_printable(cp)
                if not printable then
                        return false
                end

                if result ~= "space-separator" then
                        has_nonws = true
                end
        end

        return has_nonws
end

local function manual_unpack(what, from)
        if what[from + 1] == nil then
                return what[from]
        end

        local result = {}
        from = from or 1
        for i, item in ipairs(what) do
                if i >= from then
                        table.insert(result, item)
```

```lua
                    end
            end
            return unpack(result)
    end

    local function compare_ranges(range1, range2)
            return range1[1] < range2[1]
    end

    -- Creates a function to look up data in a module that contains "singles" (a
    -- code point-to-data map) and "ranges" (an array containing arrays that contain
    -- the low and high code points of a range and the data associated with that
    -- range).
    -- "loader" loads and returns the "singles" and "ranges" tables.
    -- "match_func" is passed the code point and either the data or the "dots", and
    -- generates the final result of the function.
    -- The varargs ("dots") describes the default data to be returned if there wasn't
    -- a match.
    -- In case the function is used more than once, "cache" saves ranges that have
    -- already been found to match, or a range whose data is the default if there
    -- was no match.
    local function memo_lookup(data_module_subpage, match_func, ...)
            local dots = { ... }
            local cache = {}
            local singles, ranges

            return function (codepoint)
                    if not singles then
                            local data_module = loader[data_module_subpage]
                            singles, ranges = data_module.singles, data_module.ranges
                    end

                    if singles[codepoint] then
                            return match_func(codepoint, singles[codepoint])
                    end

                    local range = binary_range_search(codepoint, cache)
                    if range then
                            return match_func(codepoint, manual_unpack(range, 3))
                    end

                    local range, index = binary_range_search(codepoint, ranges)
                    if range then
                            table.insert(cache, range)
                            table.sort(cache, compare_ranges)
                            return match_func(codepoint, manual_unpack(range, 3))
                    end

                    if ranges[index] then
                            local dots_range
                            if codepoint > ranges[index][2] then
                                    dots_range = {
                                            ranges[index][2] + 1,
                                            ranges[index + 1] and ranges[index + 1][1]
                                            unpack(dots)
                                    }
                            else -- codepoint < range[index][1]
                                    dots_range = {
                                            ranges[index - 1] and ranges[index - 1][2
                                            ranges[index][1] - 1,
                                            unpack(dots)
                                    }
                            end
                            table.sort(cache, compare_ranges)
```

```lua
                end

                return match_func(codepoint)
            end
    end

    -- Get a code point's combining class value in [[Module:Unicode data/combining]],
    -- and return whether this value is not zero. Zero is assigned as the default
    -- if the combining class value is not found in this data module.
    -- That is, return true if character is combining, or false if it is not.
    -- See https://www.unicode.org/reports/tr44/#Canonical_Combining_Class_Values for
    -- more information.
    p.is_combining = memo_lookup(
            "combining",
            function (codepoint, combining_class)
                    return combining_class and combining_class ~= 0 or false
            end,
            0)

    function p.add_dotted_circle(str)
            return (mw.ustring.gsub(str, ".",
                    function(char)
                            if p.is_combining(mw.ustring.codepoint(char)) then
                                    return '◌' .. char
                            end
                    end))
    end

    local lookup_control = memo_lookup(
            "control",
            function (codepoint, ccc)
                    return ccc or "assigned"
            end,
            "assigned")
    p.lookup_control = lookup_control

    function p.is_assigned(codepoint)
            return lookup_control(codepoint) ~= "unassigned"
    end

    function p.is_printable(codepoint)
            local result = lookup_control(codepoint)
            return (result == "assigned") or (result == "space-separator"), result
    end

    function p.is_whitespace(codepoint)
            local result = lookup_control(codepoint)
            return (result == "space-separator"), result
    end

    p.lookup_category = memo_lookup(
            "category",
            function (codepoint, category)
                    return category
            end,
            "Cn")

    local lookup_script = memo_lookup(
            "scripts",
            function (codepoint, script_code)
                    return script_code or 'Zzzz'
            end,
            "Zzzz")
    p.lookup_script = lookup_script
```

```lua
function p.get_best_script(str)
        -- Check type of argument, because mw.text.decode coerces numbers to stri
        require "libraryUtil".checkType("get_best_script", 1, str, "string")

        -- Convert HTML character references (including named character reference
        -- or character entities) to characters.
        str = mw.text.decode(str, true)

        local scripts = {}
        for codepoint in mw.ustring.gcodepoint(str) do
                local script = lookup_script(codepoint)

                -- Ignore "Inherited", "Undetermined", or "Uncoded" scripts.
                if not (script == "Zyyy" or script == "Zinh" or script == "Zzzz")
                        scripts[script] = true
                end
        end

        -- If scripts does not contain two or more keys,
        -- return first and only key (script code) in table.
        if not next(scripts, next(scripts)) then
                return next(scripts)
        end -- else return majority script, or else "Zzzz"?
end

function p.is_Latin(str)
        require "libraryUtil".checkType("get_best_script", 1, str, "string")
        str = mw.text.decode(str, true)

        -- Search for the leading bytes that introduce the UTF-8 encoding of the
        -- code points U+0340-U+10FFFF. If they are not found and there is at lea
        -- one Latin-script character, the string counts as Latin, because the re
        -- of the characters can only be Zyyy, Zinh, and Zzzz.
        -- The only scripts found below U+0370 (the first code point of the Greek
        -- and Coptic block) are Latn, Zyyy, Zinh, and Zzzz.
        -- See the codepage in the [[UTF-8]] article.
        if not str:find "[\205-\244]" then
                for codepoint in mw.ustring.gcodepoint(str) do
                        if lookup_script(codepoint) == "Latn" then
                                return true
                        end
                end
        end

        local Latn = false

        for codepoint in mw.ustring.gcodepoint(str) do
                local script = lookup_script(codepoint)

                if script == "Latn" then
                        Latn = true
                elseif not (script == "Zyyy" or script == "Zinh"
                                or script == "Zzzz") then
                        return false
                end
        end

        return Latn
end

-- Checks that a string contains only characters belonging to right-to-left
-- scripts, or characters of ignorable scripts.
function p.is_rtl(str)
```

```lua
        require "libraryUtil".checkType("get_best_script", 1, str, "string")
        str = mw.text.decode(str, true)

        -- Search for the leading bytes that introduce the UTF-8 encoding of the
        -- code points U+0580-U+10FFFF. If they are not found, the string can on1
        -- have characters from a left-to-right script, because the first code po
        -- in a right-to-left script is U+0591, in the Hebrew block.
        if not str:find "[\214-\244]" then
                return false
        end

        local result = false
        local rtl = loader.scripts.rtl
        for codepoint in mw.ustring.gcodepoint(str) do
                local script = lookup_script(codepoint)

                if rtl[script] then
                        result = true
                elseif not (script == "Zyyy" or script == "Zinh"
                                or script == "Zzzz") then
                        return false
                end
        end

        return result
end

local function get_codepoint(args, arg)
        local codepoint_string = args[arg]
                or errorf(2, "Parameter %s is required", tostring(arg))
        local codepoint = tonumber(codepoint_string, 16)
                or errorf(2, "Parameter %s is not a code point in hexadecimal bas
                        tostring(arg))
        if not (0 <= codepoint and codepoint <= 0x10FFFF) then
                errorf(2, "code point in parameter %s out of range", tostring(arg
        end
        return codepoint
end

local function get_func(args, arg, prefix)
        local suffix = args[arg]
                or errorf(2, "Parameter %s is required", tostring(arg))
        suffix = mw.text.trim(suffix)
        local func_name = prefix .. suffix
        local func = p[func_name]
                or errorf(2, "There is no function '%s'", func_name)
        return func
end

-- This function allows any of the "lookup" functions to be invoked. The first
-- parameter is the word after "lookup_"; the second parameter is the code point
-- in hexadecimal base.
function p.lookup(frame)
        local func = get_func(frame.args, 1, "lookup_")
        local codepoint = get_codepoint(frame.args, 2)
        local result = func(codepoint)
        if func == p.lookup_name then
                -- Prevent code point labels such as <control-0000> from being
                -- interpreted as HTML tags.
                result = result:gsub("<", "&lt;")
        end
        return result
end
```

```
function p.is(frame)
        local func = get_func(frame.args, 1, "is_")

        -- is_Latin and is_valid_pagename take strings.
        if func == p.is_Latin or func == p.is_valid_pagename or func == p.is_rtl
                return (func(frame.args[2]))
        else -- The rest take code points.
                local codepoint = get_codepoint(frame.args, 2)
                return (func(codepoint)) -- Adjust to one result.
        end
end

return p
```