Cheatography

Python String Formatting Cheat Sheet

by Gaston via cheatography.com/24027/cs/21185/

C-printf Style

General form	pf_string1 % (arg1 [, arg2 ...])
	pf_string2 % { 'name1' : arg1[, ...})
pf_string1	" <literal>|<fmt1> ... "
pf_string2	" <literal>|<fmt2> ... "
<literal>	any code point outside of fmt is output as is
%%	outputs '%' (escape)
<fmt1>	% [<conversion-flags>] [<min-field-width>] [<precision>] [<length>] <conversion-type>
<fmt2>	% "(" <key> ")" [<conversion-flags>] [<min-field-width>] [<precision>] [<length>] <conversion-type>

printf-style format specifiers

<key>	mapping key-anme in passed dictionary	
<conversion-flags>	"#"	conversion uses alternative form (see alternative form in format mini-language section)
	"0"	0 padded numeric value
	"-"	left adjust (overrides zero-padding)
	" "	leave space before positive number
	"+"	sign character "-" or "+" will preceede the converted number
<min-field-width>	"*"	minimum width read from passed value tuple and must be followed by the value itself
	<n>	minimum field width
<precision>	"." "*"	precision read from next value in value tuple followed by th evalue itself
	"." <n>	precision, <n> digits after decimal separator
<length-modifier>	"h" | "l" | "L"	ignored in Python

printf-style format specifiers (cont)

<conversion-type>	"d"|"i"	signed integer
	"o"	signed octal
	"u"	(obsolte) same as "d"
	"x"	signed hexadecimal (lower case)
	"X"	signed hexadecimal (upper case)
	"e"	floating point number in exponential form (lowercase 'e')
	"E"	floating point number in exponential form (uppercase 'E')
	"f"|"F"	decimal floating point
	"g"	floating point form,at using exponential format iof exponent less than -4 (lowercase 'e')
	"G"	floating point form,at using exponential format iof exponent less than -4 (uppercase 'e')
	"c"	single character
	"r"	string - converts any object using repr()
	"s"	string - converts any object using str()
	"a"	string - converts any object using ascii()
	"%"	literal '%'

Cheatography

Python String Formatting Cheat Sheet

by Gaston via cheatography.com/24027/cs/21185/

String format

str.format()	<format-string> "." format(<args>)	
<format string>	literal string with <field> definitions	
<args>	<arg1>, <arg2>, ...	poisitional arguments referenced by index
	<kw1> "=" <arg1>; <kw2> "=" <arg2>, ...	keyword args referenced by name
	"**" dctVar	args as dictionary, referenced by name
<field>	"{" [<field_name>] ["!" <conversion>] [":" <format_spec>] "}"	
<field_name>	<arg_name> ("." <attribute_name> | "[" <element_index> "]")*	
<arg_name>	[<digit>+ | <identifier>]	refering by index (positional arguments) or name (passed named arguments or dict variable) (see <args>)
<attribute_name>	<identifier>	
<element_index>	<digit>+ | <index_string>	
<index_string>	<any-character except "]"> +	
<conversion>	"r"	convert to string via repr()
	"s"	convert to string via str()
	"a"	convert to string via ascii()
<format-spec>	see format specification mini-language	

Formatted String (V3.6+)

"f"|"F" \" <literal>|"{{"|"}}"|<field> ... \"

<literal>	literal character	any code point except "{", "}" or NULL
"{{" or "}}"	escape	outputs "{" respectively "}"

<field>

<field>	"{" <fielddef> "}"	
<field-def>	<expr> ["!" <conversion>] [":" <format>]	
<expr>	(<conditional_expression> | " *" <or_expr>) ("," <conditional_expression> | "," "" <or_expr>)* [","] | <yield_expression>	virtually any python expressions works here. Some minor restrictions apply.
<conditional_expr>	<or_test> ["if" <or_test> "else" <expression>	
<expression>	<conditional_expr> | <lambda>	
<lambda>	python lambda expression	
<or_test>	boolean expression using "or", "and", "not" and comparisons	
<or_expr>	bitwise logical and shift expression using "|", "&", "^", "<<" and ">>"	
<conversion>	"s"	converted to string via str()
	"r"	converted to string via repr()
	"a"	converted to string via ascii()
<format>	(<literal> | NULL | <field>) *	format specifier compatible to str.format() method (see format specification mini-language below)

Format Specification Mini-Language

<format_spec>	[[<fill>]<align>] [<sign>][#][0][<width>][<grouping>][.<precision>][<type>]	
<fill>	<any character>	fill in the empty space in an aligned value
<align>	"<"	left-aliogn
	">"	right-align
	"="	numeric püadding takes place after the sign (like = padding)
	"^"	center

Format Specification Mini-Language (cont)		
<sign>	"+"	show sign for both positive and negative numbers
	"-"	show sign for negative nuimbers only (default)
	<space>	show space for positive and '-' for negative numbers
"#"	alternative form	(see section below)
"0"	"0" in front of the <width> field if no alignment is specified is wual to "0=" fill and align character combination	number padded by 0 between sign and digits
<width>	minimal field width	
<group-ing>	","	use comma for 1000 separator (V3.1+)
	"_"	use "_" (underscore) for 1000 separator (V3.6+)
<preci-sion>	number of digits after the decimal point	
<type> (integer)	"b"	binary in base-2
	"c"	character
	"d"	decimal integer
	"o"	octal integer (base-8)
	"x"	hex format (base 16) using lowercase letters
	"X"	hex format (base 16) using uppercase letters
	"n"	integer number (like "d"), using current locale for number separators
	None	same as "d"
<type> (float)	"e"	Exponent (scientific) notation using lowercase "e". Default precision is 6

Format Specification Mini-Language (cont)		
	"E"	Exponent (scientific) notation using uppercase "-E". Default precision is 6
	"f"	fixed-point notation with default precision of 6
	"F"	Same as "f" but "nan" and "inf" are displayed as "NAN" and "INF"
	"g"	general format with rounded result at given precision digit
	"G"	same as "g" except that uppercase "E", "NAN" and "INF" are displayed
	"n"	same as "g" except that it uses current locale for number separator characters
	"%"	precentage, multiplies number by 100, displays it in fixed format (like "f") and appends "%"
	None	similar to "g" buit has at least 1 digit past the decimal point
<type> (stzring)	"s"	string
	None	same as "s"

Alternative forms (#)	
for octal	"0o" preceeds octal number
for hexadecimal	"0x" preceeds hexadecimal number
floating point	always contain decimal point even if no digit follows
if precision is given	output truncated to precision