

Reference: the print() syntax:
print(objects, sep=',', end='\n')

Formatting Options

INTERPOLATION: Old Style formatting operator % - to be deprecated, most widely used, has bugs

1. string with **format**/insert (%) spec(s) % (insert values)
print("The cost of %d widgets is \$%.2f each %s." % (5, 202.95, "Ed"))
The cost of 5 widgets is \$ 202.95 each Ed.

Start of format specifier

: alternate form, 0 : zero padded,
- : left adjusted, ' ' (space) : space before
pos numbers, + : sign +/- required

Precision starts with a decimal point
followed by an integer specifying places

len mod was planned but not implemented

Note: [in brackets] means optional, % means yields or returns
There are 2 syntax formats:

2. "format string" % value to format

sft="11.4f" \ print((sft)%(-7.5129870) % -7.5130
sft is a string variable to hold the format spec statement ->
min field width (11), precision(.) 4 digits(4), floating point (f)

% [(dict key)] [conversion flags] [minimum field width [*]] [precision: .## or [*]] conversion type

Mapping key in
parens for a
dictionary value

Example of format string:
%-14.4f % left adj, min 14
char, 4 decimal places, float-
ing point or %("key1")s
use dict value for a string

integer specifying
the minimum
field width

<https://docs.python.org/2/library/stdtypes.html#string-formatting>

i/d : signed integer decimal
o : signed octal
x : signed hex lower case
X : signed hex upper case
e : flt pt exp lower case

E : flt pt exp upper case
f : floating point dec format
r : string using repr()
s : string using str()

format: **format**(string/#, **format str**) and **string format:** "{:format str}".format(string/#)

Recommended over interpolation. 2 forms: both use the same format **mini-language** (available in Python 3.5) to construct the format string. For a new user, conflicting examples can play with your head! * **format()**/**string format** are a built-in function.

format: The **format()** form is the easiest but does not support any replacement or substitution fields - only string and number

string format: {} can contain replacement fields - see pg2

format and conversion. **format**(12345.67890, '0=+20,.3f')

"{:0=+20,.3f}".format(12345.67890)

Both will yield: +000,000,012,345.679

What the **format symbols** above mean: "{: use format string (**string format**); 0 fill number with this character; = pad after sign but before number; + use a sign; 20 required width in characters; , use commas to show thousands; .3 set (3 in this case) digit precision; f number type; }" - close

How the **mini-language** statements are ordered and structured in general: (Note: **symbols must be in the order as shown below!**)

IF align used;
can be any
character

With sign: " " (space) force leading space
on + and - sign; + sign all; - neg only

forces 0 between sign &
number when no align
specified; like '0='

thousands separator (,) 000 - can't use with 'n'

width in chars

decimal places

[[fill] align] [sign] [# -alt form] [0 force padding default] [width] [_,] [.precision] [types]

<, >, ^ (centered), or =
'=' forces padding after a sign

Floats/Decimal - always have dec point; integers ->
hex, oct, binary - add 0x/0o/0b; 'g'/'G' - retain zeros

format: examples

Integer types:

b - binary
c - Unicode char
d - base 10 integer
o - Octal
x - Hex - lower cs
X - Hex - upper cs
n - like d but uses local separator definitions
Float/decimal types:
e - scientific, e - exponent
E - E for exponent
f - fixed point (default 6)
F - fixed, NAN and INF
g - general format, rounds and formats
n - like g but uses local separator definitions
% - percentage, * 100, adds "%";
None - g except 1 num > .
String: **s** - string format, can be omitted

Numeric output examples: using 12345.6789, -12345.6721, 1234, 10000000

Format string	Yields	Note " " symbol is col border	Description
"%.3f"	12345.679 -12345.672 1234.000 10000000.000		fixed, 3 places, float
"%,.2f"	12,345.68 -12,345.67 1,234.00 10,000,000.00		comma sep, fixed, 2 places, float
"%13,.2f"	12,345.68 -12,345.67 1,234.00 10,000,000.00		width=15, comma sep, fixed, 3 places, float
"^13,.0f"	12,346 -12,346 1,234 10,000,000		center, width=15, comma, fixed, 3 places, float
"^13,e"	1.234568e+04 -1.234567e+04 1.234000e+03 1.000000e+07		center, set width (15), scientific
"~^13,.0f"	~12,346~ -~12,346~ ~1,234~ ~10,000,000~		fill ~, ctr, width=15, commas, 0 dec places, float
"+,.0f"	+12,346 -12,346 +1,234 +10,000,000		sign, comma sep, fixed, no dec places, float
"=+13,.0f"	+ 12,346 - 12,346 + 1,234 + 10,000,000		pad _ after sign, sign, comma, no dec plcs, float
"%,.1%"	1,234,567.9% 1,234,567.2% 123,400.0% 1,000,000,000.0%		comma sep, 1 place, *100 & add %
	using 17 and 256 and 65534 and 65536		
"x^10d"	xxxx17xxxx xxx-256xxx xx65534xxx xx65536xxx		fill w/ x, center, width=10, integer (base 10)
">#12X"	0X11 -0X100 0XFFFE 0X10000		right align, width=12, hex (uppercase)
"b"	10001 -100000000 1111111111111111 10000000000000000		Binary conversion

string format: Ordering or Substituting text and numbers in statements

This syntax is in 3 PARTS: **Part 1** is **either** a way to identify which value is referenced by the literal or data container between the parens of .format(), for example '{0}' to select a data value, **or** a format spec designated by following the opening '{' with ':' ; for example '{:0=+20,.3f}'. **Part2** is the command - **.format()**

Part 3 is the literal strings or data containers referenced inside the .format parens. Look at it like this :

print string with {selection values} [{x} {x} ...].format(*/**- source for selection/insertion)

a string with embedded values in {} brackets holding a selection index **or** a format specification { in mini-language

.format

literal values; a **tuple** to unpack preceded by a single *; **multi-**
ple tuple items coded in the print string: [tup#[item#]]; a **dicti-**
onary to reference for keys coded in print string, preceded by **

...plus a LOT LOT more at:
www.wikipython.com

Formatting Options

Examples using the order and replacement functions of `str.format()`: string & `subs.format(subs source)`

Objects in the following examples

```
OrderString = '{1}, {0}, {2}'
StoogeTuple = ('Larry', 'Moe', 'Curley')
ShirtTuple = ('red', 'white', 'blue', 'purple')
StoogeDict = {'Straight': 'Larry',
'Dunce': 'Moe', 'Foil': 'Curley', 'Boob': 'Don'}
PetDict = {1: "cow", 2: "dog", 3: "fish"}
```

```
class Flowers(object):
    def __init__(self,
center, petals):
        self.center=center
        self.petals=petals
Daisy = Flowers
("black", "yellow")
```

Simple selection and ordering of values with **literals**
mystring = "The tourney ranking: {1}, {3}, {0}.".format /
('Larry', 'Moe', 'Curley', 'Donald')
print(mystring) # The tourney ranking is: Moe, Donald, Larry
String holding **substitution/replacement selections**
print("The tourney rank is: " + OrderString.format('Abe', 'Bob', 'Cal', 'Don'))
The tourney rank is: Bob, Abe, Cal
Named items
print("Winners: {FirstPlace}, {SecondPlace}.".format (FirstPlace =
"Bob", SecondPlace ="Don")) # Winners: Bob, Don
Use * to **unpack a single tuple (but not a list)**
print("The stooges are: {2}, {1}, and {0}.".format /
(*StoogeTuple)) # note * & sub syntax
The stooges are: Curley, Moe, and Larry.
Use the **{0[value index]}** without having to use *
print("My favorite stooge is {0[0]}.".format(StoogeTuple))
My favorite stooge is Larry.

The **[0[]]** structure enables us to select from **multiple tuples**
print("I saw {0[1]} in a {1[2]} shirt.".format(StoogeTuple, /
ShirtTuple)) # I saw Moe in a blue shirt.
Use ** to access **dictionary values by their keys**
print("The stooges are: {Straight}, {Foil}, {Dunce}.".format
(**StoogeDict)) # note ** "dictionary is external"
The stooges are: Larry, Curley, Moe.
Select a **single dictionary item**
print("My favorite stooge is {Foil}.".format(**StoogeDict))
My favorite stooge is Curley.
A **single dictionary item using the {x[]}] format and keyword**
print("One stooge is {0[Foil]}.".format(StoogeDict))
One stooge is Curley.
Select **multiple items from multiple dictionaries using keywords**
print("It look like {0[Straight]} has a {1[1]} and a /
{1[2]} ".format(StoogeDict, PetDict))
It look like Larry has a cow and a dog
Refer to an **object's attribute** # combine with a class - very powerful
print("Its petals are bright {0.petals}.".format(Daisy))
Its petals are bright yellow.
using **!r** and **!s** - example borrowed from
<https://docs.python.org/3/library/string.html#formatspec>
print("repr() shows quotes: {!r}; str() doesn't: {!s}.".format /
('test1', 'test2')) # best possible example we could imagine
repr() shows quotes: 'test1'; str() doesn't: test2

Built-in String Format Methods

.capitalize() -1st letter
.center(width[, fillchar default: space])
.ljust(width[, fillchar]) -justify
.rjust(width[, fillchar]) -right justify
.upper() -converted to uppercase
.lower() -convert to lowercase
.strip([chars]) -remove leading and trailing chars
.lstrip([chars]) -remove leading chars
.rstrip([chars]) -remove trailing chars
.title() -return a titlecased version
.zfill(width) - left fill with 0 to width
.swapcase()

Template strings: A simple substitution function imported from the **string module**. (from **string import Template**) To keep it simple: (1) use the **Template** function to build a variable with named objects preceded by \$ to be replaced with subs, (2) then use **substitute(map object, **kws)** on that variable to define replacement values and build the string. (\$\$ escapes and yields \$)

```
from string import Template
stoogeDict= {'L':"Larry", "M":"Moe", "C":"Curley"}
funnyStr= Template("$C handed the goat to $L and butted $M.")
funnyStr= funnyStr.substitute(stoogeDict)
print(funnyStr)
# Curley handed the goat to Larry and butted Moe.
....put together more suscintley
print(Template("$M and $C butted $L's goat.").substitute
(stoogeDict))
# Moe and Curley butted Larry's goat.
```

Template strings are easy,
but VERY slow to execute!

.format dates: the easy way
import datetime
d = datetime.datetime(1948, 1, 19);
print('{:%m/%d/%Y}'.format(d))
01/19/1948

New in version 3.6: f-strings - formatted string literals - prefixed with letters f or F

for more see: https://docs.python.org/3/reference/lexical_analysis.html#f-strings (2.4.3 Formatted string literals), ALSO see PEP 498

Text except }, {, or NULL - {{ & }} are
replaced with single braces

"s" | "r" | "a"
str(), repr(), ascii()

See mini-language described in **format()**
Ex: `format(string/number, 'o'+20, '3f')` explained on
side 1.

f or **F** • opening quote " • [literal text] • {replacement fields [:format string]} • [literal text] • closing quote "

f_expression : (conditional_expression | **"!r"**
or_expr (" | conditional_expression | "!" | **"!s"** or **expr**)*
[" | "] (NO BACKSLASHES IN EXPRESSION PARTS;
Must put LAMBDA in parens ())

"{" f_expression ["!" conversion] [":" format_spec] "}"
*no backslashes var!s var!r var!a - 'var' is literal variable

```
nametup = ("Larry", "Curley", "Moe") # stuff for examples
myindex, Name, width, value, x = 2, 'Curley', 12, 12345.678, 75
state, subpart, subpart2 = 'Mississippi', 'iss', 'x'
lamstate = lambda state: state if subpart in state else "unknown"
intro_string = "Money: $"
import datetime
print(f'He said his name is {nametup[myindex]}.') #use index
print(f'{Name.upper(): ^10} center & caps!') #sub and format
print(f'{intro_string}{value: {width},.2f} is cheap?') #note space
print(f'Going to {(lamstate(state)).upper(): ^20}!') #conditional
print(f'Bound for {state if subpart in state else 'unknown'}!')
print(f'Going to {state if subpart2 in state else 'unknown'}!')
print(f'Curley's IQ is about {x!r}.') #conversion example
print(f'Today is {datetime.date.today(): %m/%d/%Y}.')
```

Other notes: **formatter** - no notes, formatter module has been deprecated.
pprint module - Data pretty printer - "provides a capability to "pretty-print" arbitrary Python data structures in a form which can be used as input to the interpreter." See: <https://docs.python.org/3.6/library/pprint.html#module-pprint> Beyond the scope of this toolbox document,
but import and create object with

He said his name is Moe.
CURLEY center & caps!
Money: \$ 12,345.68 is cheap?
Going to MISSISSIPPI !
Bound for Mississippi!
Going to unknown!
Curley's IQ is about 75.
Today is 04/16/2018.

`pp = pprint.PrettyPrinter(args)`
args: indent, width, depth, stream, *, compact
then send object to output with
command: `pp.pprint(your object)`

www.wikipython.com