

Anything in `{ }` is run as Python and turned into text; after a colon comes the format spec, `{value:fill align sign width , .precision type}`. Examples use `name, n, price = "Ada", 42, 1234.5`.

PUT VALUES IN	
<code>f"Hi {name}"</code>	→ <code>'Hi Ada'</code>
<code>f"{n + 1} items"</code> any expression works	→ <code>'43 items'</code>
<code>f"{name.upper()}"</code> method calls too	→ <code>'ADA'</code>
<code>f"{name[0]}"</code> indexing and slicing	→ <code>'A.'</code>
<code>f"{'odd' if n % 2 else 'even'}"</code> inline if/else	→ <code>'even'</code>
<code>f"{[1, 2]}"</code> lists and dicts print as they look	→ <code>'[1, 2]'</code>

WIDTH AND ALIGNMENT	
<code>f"{name:<6}"</code> left: the default for text	→ <code>'[Ada]'</code>
<code>f"{name:>6}"</code> right: the default for numbers	→ <code>'[Ada]'</code>
<code>f"{name:^7}"</code> centre	→ <code>'[Ada]'</code>
<code>f"{name:*^7}"</code> fill character goes before the align	→ <code>'**Ada**'</code>
<code>f"{n:6}"</code> numbers line up on the right	→ <code>'[42]'</code>
<code>f"{-n:=6}"</code> = pads between the sign and digits	→ <code>'[- 42]'</code>
<code>f"{name:.2}"</code> precision on text cuts it short	→ <code>'Ad'</code>
<code>f"{'LoveLace':.3}..."</code> cut to fit a column	→ <code>'Lov...'</code>

DECIMALS AND BIG NUMBERS	
<code>f"{3.14159:.2f}"</code> f: fixed number of decimals	→ <code>'3.14'</code>
<code>f"{2.5:.0f}"</code> halves round to even, like round()	→ <code>'2'</code>
<code>f"{price:,.2f}"</code> thousands separator and decimals	→ <code>'1,234.50'</code>
<code>f"{1000000:_}"</code> underscores instead of commas	→ <code>'1_000_000'</code>
<code>f"{0.256:.1%}"</code> multiplies by 100 for you	→ <code>'25.6%'</code>

<code>f"{6.022e23:.3e}"</code> scientific notation	→ <code>'6.022e+23'</code>
<code>f"{0.000123:.2g}"</code> g: significant figures, picks e or f	→ <code>'0.00012'</code>
<code>f"{price:10.2f}"</code> width 10 includes the decimals	→ <code>'1234.50'</code>

SIGNS AND ZERO PADDING	
<code>f"{n:+}"</code> always show the sign	→ <code>'+42'</code>
<code>f"{n: }"</code> space: a gap where + would go	→ <code>' 42'</code>
<code>f"{7:03}"</code> 0 before the width pads with zeros	→ <code>'007'</code>
<code>f"{-7:04}"</code> zeros go after the sign	→ <code>'-007'</code>
<code>f"{3.14:07.2f}"</code> the width counts the point	→ <code>'0003.14'</code>
<code>f"INV-{n:06}"</code> fixed-length ids	→ <code>'INV-000042'</code>

BINARY, OCTAL, HEX	
<code>f"{255:x}"</code> X for upper case	→ <code>'ff'</code>
<code>f"{255:#x}"</code> the # flag adds the 0x prefix	→ <code>'0xff'</code>
<code>f"{10:b}"</code> binary; o for octal	→ <code>'1010'</code>
<code>f"{5:08b}"</code> a full byte	→ <code>'00000101'</code>
<code>f"#{255:02x}{0:02x}{128:02x}"</code> RGB to a hex colour	→ <code>'#ff0080'</code>

DATES AND TIMES	
<pre>from datetime import datetime d = datetime(2026, 3, 14, 9, 30) fmt = "%d/%m/%y"</pre>	
<code>f"{d:%Y-%m-%d}"</code> the spec goes straight to strftime	→ <code>'2026-03-14'</code>
<code>f"{d:%d %b %Y}"</code>	→ <code>'14 Mar 2026'</code>
<code>f"{d:%A, %H:%M}"</code>	→ <code>'Saturday, 09:30'</code>
<code>f"{d:%I:%M %p}"</code> 12-hour clock	→ <code>'09:30 AM'</code>
<code>f"{d:{fmt}}"</code> format kept in a variable	→ <code>'14/03/26'</code>
<code>f"{d.day}/{d.month}"</code> no zero padding, unlike %d	→ <code>'14/3'</code>

DEBUGGING	
<code>f"{n=}"</code> = shows the expression and its value	→ <code>'n=42'</code>
<code>f"{n * 2 =}"</code> spaces around = are kept	→ <code>'n * 2 = 84'</code>
<code>f"{price=:.1f}"</code> a format spec still works	→ <code>'price=1,234.5'</code>
<code>f"{name!r}"</code> !r uses repr(): shows the quotes	→ <code>"'Ada'"</code>
<code>f"{name=}"</code> = uses repr() unless you add a spec	→ <code>"name='Ada'"</code>
<code>f"{ 'café' !a}"</code> !a: ascii(), escapes non-ASCII	→ <code>"'caf\\xe9'"</code>
<code>f"{name!r:>8}"</code> conversion first, then the colon	→ <code>" 'Ada'"</code>

NESTED FIELDS	
<pre>w, p, fill = 10, 3, "*" spec = ",.2f"</pre>	
<code>f"{3.14159:{w}.{p}f}"</code> width and precision from variables	→ <code>' 3.142'</code>
<code>f"{'Ada':{fill}^{w}}"</code> fill can be a variable too	→ <code>'***Ada***'</code>
<code>f"{1234.5:{spec}}"</code> or the whole spec	→ <code>'1,234.50'</code>

BRACES, QUOTES, LINES	
<code>f"{{n}} is {n}"</code> double the brace for a literal one	→ <code>'{n} is 42'</code>
<code>f"{name.replace('a', 'o')}"</code> other quotes inside the braces	→ <code>'Ado'</code>
<code>f"{'\n'.join(['a', 'b'])}"</code> same quotes and \ allowed since 3.12	→ <code>'a\nb'</code>
<code>msg = (f"Name: {name}\n" f"Age: {n}")</code> split a long one: adjacent strings join	
<code>f"""Name: {name} Age: {n}"""</code> triple quotes keep the line breaks	
<code>rf"d{{{n}}}"</code> rf: raw f-string, handy for regex	

TABLES AND REPORTS	
<pre>rows = [("tea", 3.5), ("cake", 12.0)]</pre>	
<code>for item, cost in rows:</code> <code>print(f"item:<8}{cost:>8,.2f}")</code> fixed widths make the columns line up	
<code>f"{'Item':<8}{ 'Cost':>8}"</code> → <code>'Item Cost'</code> header: same widths as the rows	
<code>f"{'Total':.<12}{1234.5:>9,.2f}"</code> → <code>'Total..... 1,234.50'</code> dot leaders	
<code>f"{'':-^20}"</code> a rule line	→ <code>'-----'</code>

WHEN NOT TO USE AN F-STRING	
<pre>tpl = "{name} has {n}"</pre>	
<code>tpl.format(name="Ada", n=3)</code> template stored in a variable	→ <code>'Ada has 3'</code>
<code>format(1234.5, ",.2f")</code> same spec, no string around it	→ <code>'1,234.50'</code>
<code>log.info("user %s", name)</code> logging: it fills in the value only if needed	

WATCH OUT	
<code>"Hi {name}"</code> no f in front, so the braces print as typed	
<code>f"{name:.2f}"</code> number specs on text: float() it first	→ <code>raises ValueError</code>
<code>f"{2.675:.2f}"</code> floats are not exact; use Decimal for money	→ <code>'2.67'</code>
<code>f"{'LoveLace':5}"</code> width is a minimum: nothing is cut	→ <code>'LoveLace'</code>
<code>f"{n,}"</code> a comma outside the spec makes a tuple	→ <code>'(42,)'</code>
<code>f"{n:5+}"</code> the spec has a fixed order: write +5	→ <code>raises ValueError</code>
<code>f"{price:.2f}" + 1</code> TypeError: the result is text; keep the number for maths	
<code>f"{price:.2}"</code> .2 alone means 2 significant figures	→ <code>'1.2e+03'</code>