Positional formatting

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data scientist

What is string formatting?

- *String interpolation*
- Insert a custom string or variable in predefined text:

```python
custom_string = "String formatting"
print(f"{custom_string} is a powerful technique")
```

```
String formatting is a powerful technique
```

- Usage:
  - Title in a graph
  - Show message or error
  - Pass a statement to a function

Methods for formatting

- Positional formatting
- Formatted string literals
- Template method

Positional formatting

- Placeholder replace by value

'text {}'.format(value)

- `str.format()`

```python
print("Machine learning provides {} the ability to learn {}".format("systems", "automatically"))
```

```
Machine learning provides systems the ability to learn automatically
```

Positional formatting

- Use variables for both the initial string and the values passed into the method

```
my_string = "{} rely on {} datasets"
method = "Supervised algorithms"
condition = "labeled"
```

```
print(my_string.format(method, condition))
```

```
Supervised algorithms rely on labeled datasets
```

Reordering values

- Include an index number into the placeholders to reorder values

```
print("{} has a friend called {} and a sister called {}".format("Betty", "Linda", "Daisy"))
```

```
Betty has a friend called Linda and a sister called Daisy
```

```
print("{2} has a friend called {0} and a sister called {1}".format("Betty", "Linda", "Daisy"))
```

```
Daisy has a friend called Betty and a sister called Linda
```

Named placeholders

- Specify a name for the placeholders

```
tool="Unsupervised algorithms"
goal="patterns"
print("{title} try to find {aim} in the dataset".format(title=tool, aim=goal))
```

```
Unsupervised algorithms try to find patterns in the dataset
```

Named placeholders

```
my_methods = {"tool": "Unsupervised algorithms", "goal": "patterns"}
```

```
print('{data[tool]} try to find {data[goal]} in the dataset'.format(data=my_methods))
```

```
Unsupervised algorithms try to find patterns in the dataset
```

Format specifier

- Specify data type to be used: `{index:specifier}`

```
print("Only {0:f}% of the {1} produced worldwide is {2}!".format(0.5155675, "data", "analyzed"))
```

```
Only 0.515568% of the data produced worldwide is analyzed!
```

```
print("Only {0:.2f}% of the {1} produced worldwide is {2}!".format(0.5155675, "data", "analyzed"))
```

```
Only 0.52% of the data produced worldwide is analyzed!
```

Formatting datetime

```python
from datetime import datetime
print(datetime.now())
```

```
datetime.datetime(2019, 4, 11, 20, 19, 22, 58582)
```

```python
print("Today's date is {:%Y-%m-%d %H:%M}".format(datetime.now()))
```

```
Today's date is 2019-04-11 20:20
```

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

Formatted string literal

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

f-strings

- Minimal syntax
- Add prefix `f` to string

f"literal string {expression}"

```python
way = "code"
method = "learning Python faster"
print(f"Practicing how to {way} is the best method for {method}")
```

```
Practicing how to code is the best method for learning Python faster
```

Type conversion

- Allowed conversions:
    - `!s` (string version)
    - `!r` (string containing a printable representation, i.e. with quotes)
    - `!a` (some as !r but escape the non-ASCII characters)

```
name = "Python"
print(f"Python is called {name!r} due to a comedy series")
```

```
Python is called 'Python' due to a comedy series
```

Format specifiers

- Standard format specifier:
  - `e` (scientific notation, e.g. 5 10^3)
  - `d` (digit, e.g. 4)
  - `f` (float, e.g. 4.5353)

```
number = 90.41890417471841
print(f"In the last 2 years, {number:.2f}% of the data was produced worldwide!")
```

```
In the last 2 years, 90.42% of the data was produced worldwide!
```

Format specifiers

- `datetime`

```
from datetime import datetime
my_today = datetime.now()
```

```
print(f"Today's date is {my_today:%B %d, %Y}")
```

```
Today's date is April 14, 2019
```

Index lookups

```
family = {"dad": "John", "siblings": "Peter"}
```

```
print("Is your dad called {family[dad]}?".format(family=family))
```

```
Is your dad called John?
```

- Use quotes for index lookups: `family["dad"]`

```
print(f"Is your dad called {family[dad]}?")
```

```
NameError: name 'dad' is not defined
```

Escape sequences

- Escape sequences: backslashes `\`

```
print("My dad is called "John"")
```

```
SyntaxError: invalid syntax
```

```
my_string = "My dad is called \"John\""
```

```
My dad is called "John"
```

Escape sequences

```python
family = {"dad": "John", "siblings": "Peter"}
```

- Backslashes are not allowed in f-strings

```python
print(f"Is your dad called {family[\"dad\"]}?")
```

```
SyntaxError: f-string expression part cannot include a backslash
```

```python
print(f"Is your dad called {family['dad']}?")
```

```
Is your dad called John?
```

Inline operations

- Advantage: evaluate expressions and call functions inline

```
my_number = 4
my_multiplier = 7
```

```
print(f'{my_number} multiplied by {my_multiplier} is {my_number * my_multiplier}')
```

```
4 multiplied by 7 is 28
```

Calling functions

```python
def my_function(a, b):
  return a + b
```

```python
print(f"If you sum up 10 and 20 the result is {my_function(10, 20)}")
```

```
If you sum up 10 and 20 the result is 30
```

Let's practice!

REGULAR EXPRESSIONS IN PYTHON

Template method

REGULAR EXPRESSIONS IN PYTHON

Maria Eugenia Inzaugarat
Data Scientist

Template strings

- Simpler syntax
- Slower than f-strings
- Limited: don't allow format specifiers
- Good when working with externally formatted strings

Basic syntax

```python
from string import Template
my_string = Template('Data science has been called $identifier')
my_string.substitute(identifier="sexiest job of the 21st century")
```

```
'Data science has been called sexiest job of the 21st century'
```

Substitution

- Use many `$identifier`
- Use variables

```python
from string import Template
job = "Data science"
name = "sexiest job of the 21st century"
my_string = Template('$title has been called $description')
my_string.substitute(title=job, description=name)
```

```
'Data science has been called sexiest job of the 21st century'
```

Substitution

- Use `${identifier}` when valid characters follow identifier

```
my_string = Template('I find Python very ${noun}ing but my sister has lost $noun')
my_string.substitute(noun="interest")
```

```
'I find Python very interesting but my sister has lost interest'
```

Substitution

- Use `$$` to escape the dollar sign

```
my_string = Template('I paid for the Python course only $$ $price, amazing!')
my_string.substitute(price="12.50")
```

```
'I paid for the Python course only $ 12.50, amazing!'
```

Substitution

- Raise error when placeholder is missing

```
favorite = dict(flavor="chocolate")
my_string = Template('I love $flavor $cake very much')
my_string.substitute(favorite)
```

```
Traceback (most recent call last):
KeyError: 'cake'
```

Substitution

```
favorite = dict(flavor="chocolate")
my_string = Template('I love $flavor $cake very much')
```

```
try:
    my_string.substitute(favorite)
except KeyError:
      print("missing information")
```

```
missing information
```

Safe substitution

- Always tries to return a usable string
- Missing placeholders will appear in resulting string

```
favorite = dict(flavor="chocolate")
my_string = Template('I love $flavor $cake very much')
my_string.safe_substitute(favorite)
```

```
'I love chocolate $cake very much'
```

Which should I use?

- `str.format()` :
    - Good to start with. Concepts apply to f-strings.
    - Compatible with all versions of Python.
- **f-strings**:
    - Always advisable above all methods.
    - Not suitable if not working with modern versions of Python (3.6+).
- **Template strings**:
    - When working with external or user-provided strings

Let's practice!

REGULAR EXPRESSIONS IN PYTHON