Programming with Python

Konstantins Tarasjuks

Agenda for Today

Printing format - .format()

String indexing

Slicing

Basic Formatting

Simple positional formatting is probably the most common use-case. Use it if the order of your arguments is not likely to change and you only have very few elements you want to concatenate.

Since the elements are not represented by something as descriptive as a name this simple style should only be used to format a relatively small number of elements.

Old
```
'%s %s' % ('one', 'two')
```

New
```
'{} {}'.format('one', 'two')
```

Output
```
o n e   t w o
```

Old
```
'%d %d' % (1, 2)
```

New
```
'{} {}'.format(1, 2)
```

Output
```
1 2
```

New
```
'{1} {0}'.format('one', 'two')
```

Output
```
t w o   o n e
```

Padding and aligning strings

- By default values are formatted to take up only as many characters as needed to represent the content. It is however also possible to define that a value should be padded to a specific length. Unfortunately the default alignment differs between old and new style formatting. The old style defaults to right aligned while for new style it's left.

Align right:

Old

```
'%10s' % ('test',)
```

New

```
'{:>10}'.format('test')
```

Output

```
      test
```

Align left:

Old

```
'%-10s' % ('test',)
```

New

```
'{:10}'.format('test')
```

Output

```
test
```

Padding and aligning strings

- Again, new style formatting surpasses the old variant by providing more control over how values are padded and aligned.

New

```
'{:_<10}'.format('test')
```

Output

```
test______
```

Padding and aligning strings

And also center align values:

New

```
'{:^10}'.format('test')
```

Output

```
   test   
```

When using center alignment where the length of the string leads to an uneven split of the padding characters the extra character will be placed on the right side:

New

```
'{:^6}'.format('zip')
```

Output

```
 zip  
```

Numbers formatting

Integers:

Old
```
'%d' % (42,)
```

New
```
'{:d}'.format(42)
```

Output
```
4 2
```

Floats:

Old
```
'%f' % (3.141592653589793,)
```

New
```
'{:f}'.format(3.141592653589793)
```

Output
```
3 . 1 4 1 5 9 3
```

Old
```
'%4d' % (42,)
```

New
```
'{:4d}'.format(42)
```

Output
```
   4 2
```

Numbers formatting

Old

```
'%06.2f' % (3.141592653589793,)
```

New

```
'{:06.2f}'.format(3.141592653589793)
```

Output

```
0 0 3 . 1 4
```

Old

```
'%04d' % (42,)
```

New

```
'{:04d}'.format(42)
```

Output

```
0 0 4 2
```

Signed Numbers

Old
```
'%+d' % (42,)
```

New
```
'{:+d}'.format(42)
```

Output
```
+42
```

Old
```
'% d' % ((- 23),)
```

New
```
'{: d}'.format((- 23))
```

Output
```
-23
```

Old
```
'% d' % (42,)
```

New
```
'{: d}'.format(42)
```

Output
```
 42
```

New
```
'{:=5d}'.format((- 23))
```

Output
```
-  23
```

New
```
'{:=+5d}'.format(23)
```

Output
```
+  23
```

Named placeholders

Setup

```
data = {'first': 'Hodor', 'last': 'Hodor!'}
```

Old

```
'%(first)s %(last)s' % data
```

New

```
'{first} {last}'.format(**data)
```

Output

```
H o d o r   H o d o r !
```

New

```
'{first} {last}'.format(first='Hodor', last='Hodor!')
```

Output

```
H o d o r   H o d o r !
```

Named placeholders

```
person = {'first': 'Jean-Luc', 'last': 'Picard'}
```

New

```
'{p[first]} {p[last]}'.format(p=person)
```

Output

```
J e a n - L u c   P i c a r d
```

Setup

```
data = [4, 8, 15, 16, 23, 42]
```

New

```
'{d[4]} {d[5]}'.format(d=data)
```

Output

```
2 3   4 2
```

Datetime

Setup

```
from datetime import datetime
```

New

```
'{:%Y-%m-%d %H:%M}'.format(datetime(2001, 2, 3, 4, 5))
```

Output

```
2001-02-03 04:05
```

Setup

```
from datetime import datetime
dt = datetime(2001, 2, 3, 4, 5)
```

New

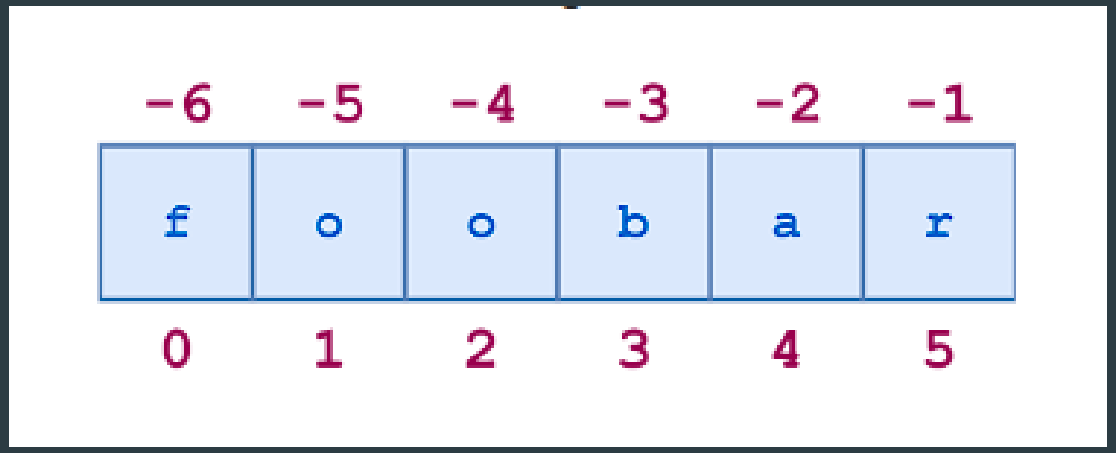
```
'{:{dfmt} {tfmt}}'.format(dt, dfmt='%Y-%m-%d', tfmt='%H:%M')
```

Output

```
2001-02-03 04:05
```

String indexing

- Often in programming languages, individual items in an ordered set of data can be accessed directly using a numeric index or key value. This process is referred to as indexing.
- In Python, strings are ordered sequences of character data, and thus can be indexed in this way. Individual characters in a string can be accessed by specifying the string name followed by a number in square brackets ([]).
- String indexing in Python is zero-based: the first character in the string has index 0, the next has index 1, and so on. The index of the last character will be the length of the string minus one.



String slice

- string_name[start:end:step]

Python

```
>>> s = 'foobar'

>>> s[0:6:2]
'foa'

>>> s[1:6:2]
'obr'
```

Python

```
>>> s = 'foobar'
>>> s[5:0:-2]
'rbo'
```

Python

```
>>> s = 'If Comrade Napoleon says it, it must be right.'
>>> s[::-1]
'.thgir eb tsum ti ,ti syas noelopaN edarmoC fI'
```

Modifying string using slicing

Python

```
>>> s = 'foobar'
>>> s[3] = 'x'
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    s[3] = 'x'
TypeError: 'str' object does not support item assignment
```

Python

```
>>> s = s[:3] + 'x' + s[4:]
>>> s
'fooxar'
```

Python

```
>>> s = 'foobar'
>>> s = s.replace('b', 'x')
>>> s
'fooxar'
```

Modifying string using slicing

- Methods list that are built in:

1. String_name.capitalize() - Capitalizes the target string.
2. String_name.lower() - Converts alphabetic characters to lowercase.
3. String_name.swapcase() - Swaps case of alphab
4. String_name.title() - Converts the target string to “title case.”
5. String_name.upper() - Converts alphabetic characters to uppercase.
6. There are a lot of more methods to center string etc.

Check reference slide for more info

Python

```
>>> 'FOO Bar 123 baz qUX'.swapcase()
'foo bAR 123 BAZ Qux'
```

Python

```
>>> "what's happened to ted's IBM stock?".title()
"What'S Happened To Ted'S Ibm Stock?"
```

Useful methods for Strings

- s.count(<sub>[, <start>[, <end>]]) - Counts occurrences of a substring in the target string.

```
Python

>>> 'foo goo moo'.count('oo')
3
```

String slice - find

`s.find(<sub>[, <start>[, <end>]])`

> Searches the target string for a given substring.

You can use `.find()` to see if a Python string contains a particular substring. `s.find(<sub>)` returns the lowest index in `s` where substring `<sub>` is found:

Python >>>

```
>>> 'foo bar foo baz foo qux'.find('foo')
0
```

This method returns `-1` if the specified substring is not found:

Python >>>

```
>>> 'foo bar foo baz foo qux'.find('grault')
-1
```

The search is restricted to the substring indicated by `<start>` and `<end>`, if they are specified:

Python >>>

```
>>> 'foo bar foo baz foo qux'.find('foo', 4)
8
>>> 'foo bar foo baz foo qux'.find('foo', 4, 7)
-1
```

Replace()

Syntax :

```
string.replace(old, new, count)
```

Parameters :

old *– old substring you want to replace.*

new *– new substring which would replace the old substring.*

count *– the number of times you want to replace the old substring with the new substring*

*(**Optional**)*

Python3

```
# Python3 program to demonstrate the
# use of replace() method

string = "geeks for geeks geeks geeks geeks"

# Prints the string by replacing all
# geeks by Geeks
print(string.replace("geeks", "Geeks"))

# Prints the string by replacing only
# 3 occurrence of Geeks
print(string.replace("geeks", "GeeksforGeeks", 3))
```

Output :

```
Geeks for Geeks Geeks Geeks Geeks
GeeksforGeeks for GeeksforGeeks GeeksforGeeks geeks geeks
```

List slice

- The slice() function returns a slice object. A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end. You can also specify the step, which allows you to e.g. slice only every other item.

Syntax

```
slice(start, end, step)
```

Parameter Values

Parameter	Description
start	Optional. An integer number specifying at which position to start the slicing. Default is 0
end	An integer number specifying at which position to end the slicing
step	Optional. An integer number specifying the step of the slicing. Default is 1

Slicing examples

```
a = ("a", "b", "c", "d", "e", "f", "g", "h")

x = slice(3, 5)

print(a[x])
```

```
('d', 'e')
```

```
a = ("a", "b", "c", "d", "e", "f", "g", "h")

x = slice(0, 8, 3)

print(a[x])
```

```
('a', 'd', 'g')
```

Task 1

- Take string - ‘My name is something must be here’
- Print out with your name using slicing – e.g. ‘My name is Konstantins’
- Print in 2 different options

1. using simple slice and your name
2. Using printing format {}

Task 2 - Cypher

- Uncypher this string ts41h2x5i2assfa2i25asfawmda5y76spfdaasa4sdczss24

Task 3

- Redo task 1 – using this logic

```
text = 'Python Programing'

# get slice object to slice Python
sliced_text = slice(6)
print(text[sliced_text])

# Output: Python
```

Task 4

- Use this ('is', 'name', 'my’) and print My name is and your name
- Use slices and String .format()

Task 5

- Uncypher this using find and replace command 't1234h1234i1234saaaaiaaaasbbbbm1234yaaaapbbbbabbbbsaaaas1234'

Reference

- Format command - https://pyformat.info/
- String indexing - https://tinyurl.com/4kkxms59
- String replace - https://www.geeksforgeeks.org/python-string-replace/
- Slicing - https://tinyurl.com/yc2mcfab


THANK YOU FOR YOUR ATTENTION!
...YOU ALWAYS HAVE MY ATTENTION.
makeameme.org