Strings

String is one of the most commonly used data type in Python. Strings in Python are objects.

1. String Objects

1.1 Create Strings

Strings are enclosed in either single, double-quotes or triple double-quotes.

In [ ]:

```
x = 'hi'
y = "hello"
z = """world"""
```

In [ ]:

```
m = "I'm here"
print(m)
```

Triple double-quotes are commonly used to initialize multi-line strings.

In [ ]:

```
z = """How
are
you?"""
z
```

String Length

Use `len()` function to get the length of a string.

In [ ]:

```
len('hello')
```

In [ ]:

```
len([1,2,3])
```

1.2 Immutable

String is immutable object in Python.

- When a string variable is assigned with a new value, the variable is actually pointing to a new memory location where the new value is stored.

In following example, we use `id()` to check memory locations of x before and after value change.

In [ ]:

```
x = 'hi'
print(id(x))
x = 'hello'
print(id(x))
```

In [ ]:

```
x[0] = 'H'
```

2. Indexing and Slicing

2.1 Indexing

A string is a collection of characters. Each character can be accessed by its index (location).

In Python, index value starts from 0.

In [ ]:

```
x = 'python-string'
print(x[0])
x[2]
```

Negative Indexing

Python also supports negative indexes.

- Index of '-1' represents the last character of the String.

In [ ]:

```
print(x[-1])
print(x[-2])
```

1.2 Slicing

Slicing is to retrieve a range of characters in a String.

The `x[a:b]` returns all characters in `x` from index `a` to `b-1`.

- Note: Character at index 'b' is not included in output.

In [ ]:

```
x = "HELLO"
print(x[0:3])
x[:3]
```

The `x[:b]` returns all characters before index `b`.

- Start-index will be set to 0 if it is ommited.
- Character at `b` is not included.

In [ ]:

```
x[:6]
```

The x[a:] returns all characters from index `a`.

- End-index will be length of `x` if it is ommited.
- Charater at `a` is included.

In [ ]:

```
x[3:]
```

You can combine positive and negative indexing in slicing.

Question: What will be the output of following code?

In [ ]:

```
x[1:-1]
```

Question: Can you modify a string by changing one of its character?

In [ ]:

```
x[0] = 'P'
```

2. String Operators

Some operators can also be applied to String objects.

2.1 Operators

Concatenation `+`

It combines two strings into one.

In [ ]:

```
x = 'Hello' + 'World'
x
```

In [ ]:

```
x = 'Five' + str(123)
x
```

Repetition `*`

It creates a new string by repeating it a given number of times.

In [ ]:

```
x = 'Hello '
x * 3
```

2.2 Membership

Exists `in`

The `in` operator returns `True` if a string (including character) is present in another string, otherwise it returns `False`.

In [ ]:

```
'hell' in 'hello'
```

Not Exists `not in`

This is a combination of `not` and `in` operators. It returns `True` if a string is NOT present in another string.

In [ ]:

```
'hell' not in 'world'
```

3. String Formatting

Following are the few ways to format string in Python.

- Operator %
- Function format()
- f-string

3.1 Operator %

It uses `%` operator to format string.

- This is the old style of string formatting in Python, which you may see it in existing code.

You need to postfix `%` with a **type-specific char** to indicate the type of data to print.

- It may cause TypeError if you use wrong type char.

Conversion	Meaning
'd', 'i'	Signed integer decimal
'o'	Signed octat value
'x', 'X'	Signed hexadecimal (lowercase, uppercase)
'e', 'E'	Floating point exponential format (lowercase, uppercase)
'f', 'F'	Floating point decimal
'r', 'a'	String (converts any Python object using repr())
's'	String (converts any Python object using str())
'%%'	No argument is converted. Result is '%

In [ ]:

```
name = 'Mark'
age = 18
y = '%s is %d years old' % (name, age)
print(y)
```

Alignment

You can allocate spaces to display text. By default, the text is aligned right in the space.

In [ ]:

```
'Aligned: %20s' % ('hi')
```

In [ ]:

```
'Aligned: %2s' % ('mark')
```

In [ ]:

```
name = 'Mark'
age = 18
```

```
print('Name: %10s' % name)
print('Age:  %10d' % age)
```

3.2 Function format()

The `format()` function in statement `'{}'.format(param)` replaces `{}` with input parameter.

- It is the recommended string formatting tool.
- It could handle more data type than printf-style (%) method
- It automatically handle data conversion for some data types.

In [ ]:

```
help(str.format)
```

In [ ]:

```
'hi {}'.format('there')
```

It can handle multiple parameters.

- Number of input parameters must be more than number of `{}`

In [ ]:

```
name = 'Mark'
age = 18
'{} is {} years old'.format(name, age)
```

In [ ]:

```
'{0} is {1} years old'.format(name, age)
```

Alignment

You can specify the size of a placeholder, and align value inside the placeholder.

- `<` align left
- `^` align centre
- `>` align right

In [ ]:

```
'Hi{:<20}'.format('World')
```

In [ ]:

```
'Hi{:^20}'.format('World')
```

In [ ]:

```
'Hi{:>20}'.format('World')
```

Positioning

Index numbers, e.g. {0}, can be used to put arguments at correct placeholders.

In [ ]:

```
'{1} from {0}'.format('SG', 'Hi')
```

Padding

By default, space is used for padding during alignment. You can also use any other character.

- Text before `:` is used for position
- Text after `:` is used for formatting

In [ ]:

```
'Hi{:#<20}'.format('World')
```

In [ ]:

```
'Hi{:#>20}'.format('World')
```

Exercise: Format a number `123` into '000123'.

In [ ]:

```
'{:0>6}'.format(123)
```

Number Formatting

- Add separator for to number
- Format a number as binary
- Specify number of decimal points

In [ ]:

```
'{:,}'.format(10000)
```

In [ ]:

```
print('{0:x}'.format(10))
print('{0:o}'.format(10))
'{0:b}'.format(10)
```

Number of decimal points. Rounding applies if necessary.

In [ ]:

```
'{:.3f}'.format(3.1415)
```

Number of significant figures. Rounding applies if necessary.

In [ ]:

```
x = 20
y = '{:0>' + str(x) + '.3}'
# y = '{:0>%d.3}' % x
y.format(3.1415)

# '{:0>10.3}'.format(3.1455)
```

Handle Lists and Dictionaries

Function format() can also use indexing to access elements in Lists or Dictionaries.

In [ ]:

```
codes = ['C', 'Java', 'Python']
'{0[2]} is good'.format(codes)
```

Note: It is `0[a]` *instead of* `0['a']`

In [ ]:

```
fruits = {'a':'apple', 'b':'banana', 'c':'cherry'}
'A: {0[a]:>10}'.format(fruits)
'B: {0[b]:>10}'.format(fruits)
'C: {0[c]:>10}'.format(fruits)

# 'C: {0["c"]:>10}'.format(fruits) # Wrong
```

In [ ]:

```
fruits['a']
# fruits[b]  # Wrong
```

3.3 Using f-string

F-strings are a new method for formatting strings in Python, which is introduced in version 3.6.

It tries to simplify the string substitution with a minimal syntax.

- A f-string starts with `f`
- Value in placeholder can be a variable, an object, or an **expression**.

```
f'Hi {what}'
```

In [ ]:

```
x = 1
f'Circle: radius {x}, area {x * x * 3.14}'
## Same as following
# 'Circle: radius {}, area {}'.format(x, x * x * 3.14)
```

In [ ]:

```
x = 2
eval('x * x * 3.14')
```

Number Formatting

Similar to formt() function, f-string handles number formatting well.

In [ ]:

```
pi = 3.1415
f'Pi: {pi:.2f}'
```

In [ ]:

```
x = 10000
f'{x:,}'
```

In [ ]:

```
x = 10
f'{x:b}'
```

Working with Objects

One distinct advantage of f-string is its ability to handle expression. This allows it to work with objects like classes, modules, functions.

In [ ]:

```
def avg(x, y):
    return (x+y)/2

x=4
y=8
f'{x} + {y} = {avg(x,y)}'
```

In [ ]:

```
'{} + {} = {}'.format(x, y, avg(x,y))
```

4. Built-in String Functions

String object comes with a number of built-in functions.

Note: All string methods returns new values. They do not change the original string.

4.1 Conversion

Method	Description
lower()	Converts a string into lower case
upper()	Converts a string into upper case
title()	Converts the first character of each word to upper case

In [ ]:

```
t = 'HELLO WORLD'.title()
u = 'hello world'.upper()
l = 'hello world'.lower()
print(t)
print(u)
print(l)
```

4.2 Comparison

Alphabet or Number

Method	Description
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isalnum()	Returns True if all characters in the string are alphanumeric

In [ ]:

```
'1234'.isalpha()
```

In [ ]:

```
'1234'.isdecimal()
```

Apart from `isdecimal()`, there are also `isdigit()` and `isnumeric()`. There are slightly difference among these functions. You can find more info at https://www.includehelp.com/python/difference-between-string-isdecimal-isdigit-isnumeric-and-methods.aspx

Starts with & Ends with

Function `startswith()` and `endswith()` can be used to check whether string starts or ends with a specific value.

In [ ]:

```
'Hello world'.startswith('hello')
```

4.3 Search

Method	Description
count()	Returns the number of times a specified value occurs in a string
find()	Searches the string for a specified value and returns its **first** position. Return -1 if not found.
rfind()	Searches the string for a specified value and returns its **last** position. Return -1 if not found.

In [ ]:

```
s = 'Today is a good day'
s.count('day')
```

In [ ]:

```
s.count('Day')
```

In [ ]:

```
s = 'Today is a good day'
s.find('day')
```

In [ ]:

```
s.rfind('day')
```

In [ ]:

```
s.find('Day')
```

4.4 Modification

Padding

By default, the padding uses a space. Otherwise use `fillchar` parameter to specify the filler character.

Method	Description
ljust()	Returns a left justified version of the string
rjust()	Returns a right justified version of the string

Method	Description
center()	Returns a centered string

In [ ]:

```
'hello'.rjust(10)
```

In [ ]:

```
'hello'.rjust(10,'#')
```

Strip

Method	Description
lstrip()	Returns a left trim version of the string
rstrip()	Returns a right trim version of the string
strip()	Returns a trimmed version of the string

By default, strip functions remove white space charcters from the beginning or the end. You can also specify one or more characters (in a string) to be stripped.

In [ ]:

```
x = '   hello world   '
x.lstrip()
```

In [ ]:

```
x = '   hello world   '
x.strip()
```

In [ ]:

```
y = '   hello*#*#*#'
y.strip('# *')
```

Question:

How to convert string `*@*  hello *##*#` into `hello`?

In [ ]:

```
'*@*  hello *##*#'.strip('*@# ')
```

Replace, Split & Join

In [ ]:

```
'Have a good day!'.replace('good', 'great')
```

The `split()` function splits the string at the specified separator, and returns a list.

- By default, the separator is a space character.

The `join()` function joins the elements in a list using a separator.

In [ ]:

```
x = 'how are you today'
y = x.split()
```

```
y
```

In [ ]:

```
' '.join(y)
```

The `split()` function also accepts other delimiter.

In [ ]:

```
y = '123-456-789'.split('-')
y
```

In [ ]:

```
'-'.join(y)
```

Reference

- https://www.techbeamers.com/python-format-string-list-dict/
- https://www.techbeamers.com/python-strings-functions-and-examples/