Sharif University of Technology
Department of Computer Engineering

Fundamentals of Programming

Python Language

Arman Malekzadeh
PhD Candidate in Artificial Intelligence

Table of contents

Strings (Recap)

Negative Indexing

- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.

```
>>> a = "Hello, World!"
>>> print(a[-1])
!
>>> print(a[-5:-2])
orl
```

Upper and Lower in Strings in Python

- Python has a set of built-in methods that you can use on strings.
- The **upper()** method returns the string in upper case.
- The **lower()** method returns the string in lower case.

```
>>> a = "Hello, World!"
>>> print(a.upper())
HELLO, WORLD!
>>> print(a.lower())
hello, world!
```

Replace String in Python

- The **replace()** method replaces a string with another string.
- The **replace()** method returns a string where a specified value is replaced with a specified value.

```
>>> a = "Hello, World!"
>>> print(a.replace("H", "J"))
Jello, World!
```

Split String in Python

- The **split()** method splits a string into a list.
- You can specify the separator, default separator is any whitespace.

```
>>> a = "Hello, World!"
>>> print(a.split(",")) # returns ['Hello', ' World!']
```

String Concatenation

- To concatenate, or combine, two strings you can use the + operator.
- To add a space between them, add a " ":

```
>>> a = "Hello"
>>> b = "World"
>>> c = a + b
>>> print(c)
HelloWorld
>>> c = a + " " + b
>>> print(c)
Hello World
```

String Format

- As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

```
>>> age = 36
>>> txt = "My name is John, I am " + age
>>> print(txt)
TypeError: can only concatenate str (not "int") to str
```

String Format

- But we can combine strings and numbers by using the **format()** method!

- The **format()** method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

```
>>> age = 36
>>> txt = "My name is John, and I am {}"
>>> print(txt.format(age))
My name is John, and I am 36
```

String Format

- The **format()** method takes unlimited number of arguments, and are placed into the respective placeholders:

```
>>> quantity = 3
>>> itemno = 567
>>> price = 49.95
>>> myorder = "I want {} pieces of item {} for {} dollars."
>>> print(myorder.format(quantity, itemno, price))
I want 3 pieces of item 567 for 49.95 dollars.
```

String Format

- You can use index numbers (a number inside the curly brackets {}) to be sure the values are placed in the correct placeholders:

```
>>> quantity = 3
>>> itemno = 567
>>> price = 49.95
>>> myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
>>> print(myorder.format(quantity, itemno, price))
I want to pay 49.95 dollars for 3 pieces of item 567.
```

Escape Characters

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash \ followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

```
>>> txt = "We are the so-called "Vikings" from the north."
SyntaxError: invalid syntax
```

Escape Characters

- To fix this problem, use the escape character \":

```
>>> txt = "We are the so-called \"Vikings\" from the north."
>>> print(txt)
We are the so-called "Vikings" from the north.
```

Escape Characters

- Other escape characters used in Python:

```
\'   Single Quote
\\   Backslash
\n   New Line
\r   Carriage Return
\t   Tab
\b   Backspace
\f   Form Feed
\ooo   Octal value
\xhh   Hex value
```

String Methods

- The **strip()** method removes any whitespace from the beginning or the end:

```
>>> a = " Hello, World! "
>>> print(a.strip()) # returns "Hello, World!"
```

String Methods

- The **len()** method returns the length of a string:

```
>>> a = "Hello, World!"
>>> print(len(a))
13
```

String Methods

- The **replace()** method replaces a string with another string:

```
>>> a = "Hello, World!"
>>> print(a.replace("H", "J"))
Jello, World!
```

String Methods

- The **split()** method splits the string into substrings if it finds instances of the separator:

```
>>> a = "Hello, World!"
>>> print(a.split(",")) # returns ['Hello', ' World!']
```

String Methods

- Check String:

```
>>> txt = "The rain in Spain stays mainly in the plain"
>>> x = "ain" in txt
>>> print(x)
True
>>> x = "ain" not in txt
>>> print(x)
False
```

String Methods in Python: `isalpha` and `isdigit`

```
>>> a = "Hello"
>>> print(a.isalpha())
True
>>> a = "Hello world!"
>>> print(a.isalpha())
False
>>> a = "123"
>>> print(a.isdigit())
True
>>> a = "123a"
>>> print(a.isdigit())
False
```

String Methods in Python: `find`

- The `find` method returns the index of the first occurrence of the substring (if found). If not found, it returns -1.

- The syntax of `find` method is:

```
str.find(sub[, start[, end]])
```

- The `find` method takes maximum of three parameters:

  - `sub` - It's the substring to be searched in the str string.
  - `start` and `end` (optional) - substring is searched within str[start:end]

String Methods in Python: `find` Example

```
>>> str = "this is string example....wow!!!"
>>> str.find("is")
2
>>> str.find("is", 14)
-1
>>> str.find("is", 5, 10)
5
>>> str.find("is", 5, 15)
5
```

String Methods in Python: `join`

- The `join` method returns a string concatenated with the elements of an iterable.

- The syntax of `join` method is:

```
string.join(iterable)
```

- The `join` method takes an iterable - objects capable of returning its members one at a time. Some examples are List, Tuple, String, Dictionary and Set

String Methods in Python: `join` Example

```
>>> numList = ['1', '2', '3', '4']
>>> separator = ', '
>>> print(separator.join(numList))
1, 2, 3, 4
>>> print(separator.join(numList).split(','))
['1', '2', '3', '4']
```

Regular Expressions

Regular Expressions in Python

- A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.
- Regular expressions are widely used in UNIX world.
- The module `re` provides full support for Perl-like regular expressions in Python.

```
>>> import re
```

Regex in Python: Symbols

- `[]` is used to indicate a set of characters.

```
>>> re.findall("[a-z]", "Hello, World!")
['e', 'l', 'l', 'o', 'o', 'r', 'l', 'd']
```

- `+` is used to indicate one or more occurrences of the pattern left to it.

```
>>> re.findall("[a-z]+", "Hello, World!")
['ello', 'orld']
```

Regex in Python: Symbols

- `.` is used to match any character (except newline).

```
>>> re.findall("...", "Hello, World!")
['Hel', 'lo,', ' Wo', 'rld']
```

- `^` is used to match the start of a string.

```
>>> re.findall("^H.", "Hello, World!")
['He']
```

Regex in Python: Symbols

- `$` is used to match the end of a string.

```
>>> re.findall("..$", "Hello, World!")
['d!']
```

- `*` is used to match zero or more occurrences of the pattern left to it.

```
>>> re.findall("..ll*o", "Hello, World!")
['Hello']
```

Regex in Python: Symbols

- `?` is used to match zero or one occurrence of the pattern left to it.

```
>>> re.findall("W.r?ld", "Hello, World!")
['World']
```

- `{}` is used to specify the number of repetitions of a pattern.

```
>>> re.findall(".{2}o", "Hello, World!")
['llo', 'Wo']
```

Regex in Python: Symbols

- [something] can be used to match any character except the ones present in the word something

```
>>> re.findall("[^abcd]{5}", "Hello, World!")
['Hello', ', Wor']
```

- Another example

```
>>> re.findall("[^!,]{3}", "Hello, World!")
['Hel', ' Wo', 'rld']
```

Regex in Python: Groups

- `|` is used to match zero or one occurrence of the pattern left to it.

```
>>> re.findall("[A-Z][a-z]{3}o|[A-Z]o", "Hello, World!")
['Hello', 'Wo']
```

- `()` is used to group sub-patterns.

```
>>> re.findall("([A-Za-z]{3}|[0-9])|([?!])", "1. Hey, are you there?!")
[('1', ''),('Hey', ''), ('are', ''), ('you', ''),
 ('the', ''), ('', '?'), ('', '!')]
```

Regex in Python: `search` function

- The `search` function returns a match object on success, None on failure.
- The syntax of `search` function is:

```
re.search(pattern, string, flags=0)
```

- The `search` function takes three parameters:
  - `pattern` - regular expression to be matched.
  - `string` - the string which would be searched to match the pattern anywhere in the string.
  - `flags` (optional) - flags to control regular expression matching.

Regex in Python: `search` function example

```
>>> import re
>>> result = re.search(r'[A-Z][a-z]{5}', 'Welcome to the Python Class')
>>> print(result.group(0))
Welcom
>>> result.start()
0
>>> result.end()
6
```

Regex in Python: `split` function

- The `split` function returns a list where the string has been split at each match.
- The syntax of `split` function is:

```
re.split(pattern, string, maxsplit=0, flags=0)
```

- The `split` function takes four parameters:
  - `pattern` - regular expression to be matched.
  - `string` - the string which would be split by the pattern.
  - `maxsplit` (optional) - The maxsplit defines the maximum number of splits.
  - `flags` (optional) - flags to control regular expression matching.

Regex in Python: `split` function example

```
>>> import re
>>> result = re.split(r'\s', 'Welcome to the Python Class')
>>> print(result)
['Welcome', 'to', 'the', 'Python', 'Class']
```

Regex in Python: `sub` function

- The `sub` function returns a string where matched occurrences are replaced with the content of replace variable.

- The syntax of `sub` function is:

```
re.sub(pattern, replace, string, count=0, flags=0)
```

- The `sub` function takes five parameters:
  - `pattern` - regular expression to be matched.
  - `replace` - content which would replace matched pattern.
  - `string` - the string which would be searched to match the pattern anywhere in the string.
  - `count` (optional) - The maxsplit defines the maximum number of splits.
  - `flags` (optional) - flags to control regular expression matching.

Regex in Python: `sub` function example

```
>>> import re
>>> result = re.sub(r'\s', '-', 'Welcome to the Python Class')
>>> print(result)
Welcome-to-the-Python-Class
```

Regex in Python: `compile` function

- The `compile` function returns a RegexObject.
- The syntax of `compile` function is:

```
re.compile(pattern, flags=0)
```

- The `compile` function takes two parameters:
  - `pattern` - regular expression to be matched.
  - `flags` (optional) - flags to control regular expression matching.

Regex in Python: `compile` function example

```
>>> import re
>>> pattern = re.compile(r'\s')
>>> result = pattern.split('Welcome to the Python Class')
>>> print(result)
['Welcome', 'to', 'the', 'Python', 'Class']
```

References

References I

[1] B Downey, A. (2015). Think Python: How to Think Like a Computer Scientist-2nd Edition.

[2] Deitel, H. M., & Deitel, P. J. (2004). C: How to program. Pearson Educacion.

Sharif University of Technology
Department of Computer Engineering

Arman Malekzadeh

Fundamentals of Programming

Python Language