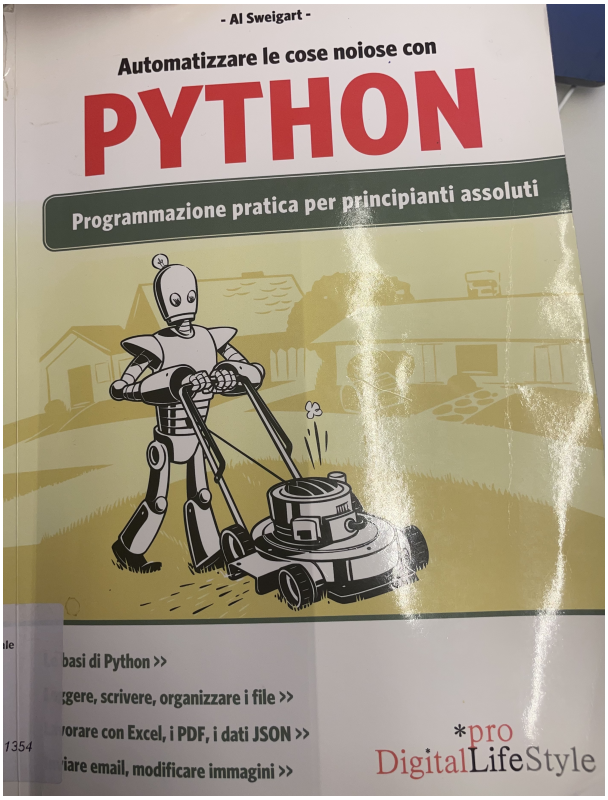
Python

Prepared by Simone Capodivento



March 8, 2025

Contents

1 Functions

```
def hello():
    print('Hello')
    print('Hello!!!')
    print('Hello to you.')

hello()
hello()
hello()  # Function calls
```

```
def hello(name):
    print('Hello ' + name)  # name is a parameter

hello('Alice')
hello('Bob')
```

1.1 Statement return

```
import random

def getAnswer(answerNumber):
    if answerNumber == 1:
        return 'It is safe'
    elif answerNumber == 2:
        return 'Exactly'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Uncertain answer, try again'
```

```
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Focus and ask again'
    elif answerNumber == 7:
        return 'My answer is no'
    elif answerNumber == 8:
        return 'The outlook is not very good'
    elif answerNumber == 9:
        return 'Very doubtful'

r = random.randint(1, 9)  # Generate a random number between 1 and 9
fortune = getAnswer(r)  # Call the function with the generated number
print(fortune)  # Print the fortune message
```

Python Code Examples

Basic Print Statements

```
print('Hello')
print('World')
```

Print with End Parameter

```
print('Hello', end="")
print('World')
```

Print with Multiple Arguments

```
>>> print('cats', 'dogs', 'mice')
    cats dogs mice
>>> print('cats', 'dogs', 'mice', sep=',')
    cats,dogs,mice
```

Scope and Variable Visibility

Local Variables

Local variables cannot access variables from other local scopes. A new local scope is created every time a function is called.

Global Variables

Global variables can only be accessed within the global scope.

Lists

A list is a value that contains one or more values in a determined sequence. The elements inside the list are called items, and they are comma-separated.

```
['cat', 'bat', 'mouse', 'elephant']
```

The positions of these elements are called indexes.

Examples

```
spam = ['cat', 'bat', 'mouse', 'elephant']
spam[0]
'cat'

spam[1]
'bat'

spam[2]
'mouse'

'hello' + spam[0]
'hellocat'

'hello ' + spam[0]
'hello cat'

spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
spam[0]
['cat', 'bat']
```

Negative Indexes

```
spam = ['cat', 'bat', 'mouse', 'lion']
spam[-1]
'lion'
```

Sublists and Slices

```
spam[1:4]
```

Working with Lists

Flexibility of Manipulation

```
catNames = []
while True:
    print('Enter the cat name ' + str(len(catNames) + 1) +
          ' (or press enter to finish):')
    name = input()
    if name == '':
        break
    catNames = catNames + [name]  # list concatenation
    print('The cat names are:')
    for name in catNames:
        print(' ' + name)
```

For Loops with Lists

```
for i in range(len(supplies)):
    print('Index ' + str(i) + ' in supplies is: ' + str(supplies[i]))

# Example output:
```

```
index 0 in supplies is: pens
index 1 in supplies is: staplers
index 2 in supplies is: flamethrower
index 3 in supplies is: binders
```

in and not in Operators

The in and not in operators are used to check whether a value exists in a list.

```
supplies = ['pens', 'staplers', 'flamethrower', 'binders']

'pens' in supplies
# True

'notebook' in supplies
# False

'staplers' not in supplies
# False
```

2 In and not in Operators

```
'howdy' in ['hello', 'hi', 'howdy', 'heyas']
# True

'howdy' not in spam
# False

'cat' not in spam
# True
```

```
myPets = ['Zophie', 'Pooka', 'Fat-tail']
print('Enter the name of an animal:')
name = input()
if name not in myPets:
    print('I don\'t have a pet named ' + name)
else:
    print(name + ' is one of my pets.')
```

3 += Operator

```
spam = 'Hello'
spam += ' world!'

spam
# 'Hello world!'
```

```
spam = ['hello', 'hi', 'howdy', 'heyas']
spam.index('hello')
# 0
```

4 append() and insert() to Add Values to Lists

```
spam = ['cat', 'dog', 'bat']
spam.append('moose')
```

5 Remove Values from Lists with `remove()`

```
spam = ['cat', 'dog', 'bat']
spam.remove('bat')
```

6 `sort()`

```
spam.sort(reverse=True)

spam
# ['bat', 'cat', 'dog']
```

7 Using `str.lower`

```
spam = ['a', 'z', 'A', 'Z']
spam.sort(key=str.lower)

spam
# ['a', 'A', 'z', 'Z']
```

8 Tuple

```
eggs = ('hello', 42, 0.5)

eggs[0]
# 'hello'

eggs[1:3]
# (42, 0.5)

len(eggs)
# 3
```

9 Conversion Type Functions: `list()` and `tuple()`

```
tuple = ['cat', 'dog', 5]

tuple = ('cat', 'dog', 5)
```

10 Reference Assignment

```
spam = 42
cheese = spam
spam = 100

spam
# 100
```

```
spam = [0, 1, 2, 3, 4, 5]

cheese = spam

cheese[1] = 'Hello!'

spam
# [0, 'Hello!', 2, 3, 4, 5]

cheese
# [0, 'Hello!', 2, 3, 4, 5]
```

11 Passing References

```
def eggs(someParameter):
    someParameter.append('Hello')

spam = [1, 2, 3]
eggs(spam)

print(spam)
# [1, 2, 3, 'Hello']
```

12 `copy()` and `deepcopy()`

```
import copy
spam = ['A', 'B', 'C', 'D']
cheese = copy.copy(spam)
cheese[1] = 42

spam
# ['A', 'B', 'C', 'D']

cheese
# ['A', 42, 'C', 'D']
```

If the list you need to copy contains lists, use `copy.deepcopy()` instead of `copy.copy()`. Type of dictionaries The index of dictionaries is called **keys**, and they consist of key-value pairs.

```
myCat = {'size': 'fat', 'color': 'gray', 'temperament': 'cheerful'}
```

Dictionaries and lists

```
pets = {'cat': ['Zophie', 'Pooka'], 'dog': ['Rex', 'Buddy']}
print(pets['cat'])
# ['Zophie', 'Pooka']
```

Python Dictionaries and Lists

Dictionaries store key-value pairs and do not maintain a specific order. Below are essential methods for working with them.

Accessing Keys, Values, and Items

```
spam = {'color': 'red', 'age': 42}

for v in spam.values():
    print(v)

for k, v in spam.items():
    print(f'Key: {k} Value: {v}')
# Output:
# Key: color Value: red
# Key: age Value: 42
```

Retrieving Values with `get()`

`get()` returns the value of a key or a default if the key is missing.

```
spam = {'name': 'Pooka', 'age': 5}
color = spam.get('color', 'unknown')
print(color)  # Output: unknown
```

Setting Default Values with `setdefault()`

`setdefault()` assigns a value to a key only if it is absent.

```
spam = {'name': 'Pooka', 'age': 5}
spam.setdefault('color', 'black')
print(spam)  # Output: {'name': 'Pooka', 'age': 5, 'color': 'black'}
```

Character Frequency

The following Python code analyzes the frequency of characters in a string:

```
message = 'It was a cold March day, and the bells were ringing at noon.'
count = {}

for character in message:
    count.setdefault(character, 0)
    count[character] += 1

print(count)
```

The generated output is:

```
{' ': 14,
 ',': 1,
 '.': 1,
 'I': 1,
 'M': 1,
 'a': 9,
 'b': 1,
 'c': 1,
 'd': 5,
 'e': 7,
 'g': 3,
 'h': 2,
 'i': 5,
 'l': 3,
 'n': 6,
 'o': 5,
 'r': 6,
```

```
 's': 4,
 't': 7,
 'w': 2,
 'y': 2}
```

Pretty Printing with `pprint()`

You can improve the presentation of the output by using the `pprint()` method from the `pprint` module:

```
import pprint
message = 'It was a cold March day, and the bells were ringing at noon.'
count = {}
for character in message:
    count.setdefault(character, 0)
    count[character] += 1
pprint.pprint(count)
```

The generated output is:

```
{' ': 14,
 ',': 1,
 '.': 1,
 'I': 1,
 'M': 1,
 'a': 9,
 'b': 1,
 'c': 1,
 'd': 5,
 'e': 7,
 'g': 3,
 'h': 2,
 'i': 5,
 'l': 3,
 'n': 6,
 'o': 5,
 'r': 6,
 's': 4,
 't': 7,
 'w': 2,
 'y': 2}
```

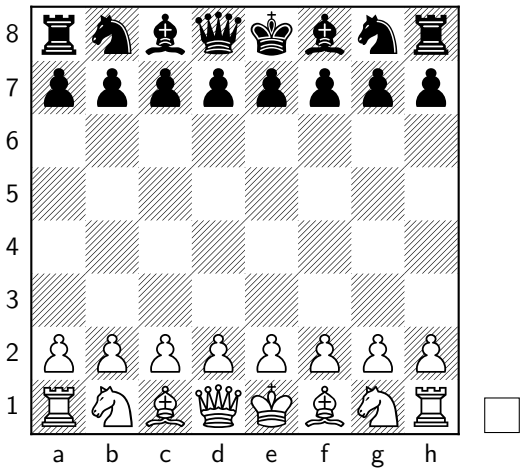
Representing a Chessboard in Python

In Python, a chessboard can be represented as a 2D list or dictionary. Each square can hold a piece, like 'K' for King, 'Q' for Queen, 'P' for Pawn, etc. Here's an example of how you might represent a chessboard in Python using a dictionary:

```
# Chessboard setup
white_pieces = {
    'a1': 'R', 'b1': 'N', 'c1': 'B', 'd1': 'Q', 'e1': 'K', 'f1': 'B', 'g1': 'N'
        , 'h1': 'R',
    'a2': 'P', 'b2': 'P', 'c2': 'P', 'd2': 'P', 'e2': 'P', 'f2': 'P', 'g2': 'P'
        , 'h2': 'P'
}

black_pieces = {
    'a7': 'p', 'b7': 'p', 'c7': 'p', 'd7': 'p', 'e7': 'p', 'f7': 'p', 'g7': 'p'
        , 'h7': 'p',
    'a8': 'r', 'b8': 'n', 'c8': 'b', 'd8': 'q', 'e8': 'k', 'f8': 'b', 'g8': 'n'
        , 'h8': 'r'
}
```

```
chessboard = {**white_pieces, **black_pieces}
```



In this case, the chessboard is represented as a dictionary, where the keys are the coordinates of each square (like 'a1', 'b1', etc.) and the values are the pieces on that square. Uppercase letters represent white pieces, and lowercase letters represent black pieces.

To display the board, you can loop through the dictionary and print the pieces in the format of a traditional chessboard:

```
def print_chessboard(board):
    for rank in range(8, 0, -1):
        row = ''
        for file in 'abcdefgh':
            row += board.get(f'{file}{rank}', '.') + ' '
        print(row.strip())

# Print the chessboard
print_chessboard(chessboard)
```

This will output a chessboard with the pieces in their starting positions.

Tic-Tac-Toe in Python

Here is an example of a simple Tic-Tac-Toe game implemented in Python. The board is represented as a dictionary, and the game allows two players to take turns making moves.

```
theBoard = {'top-L': ' ', 'top-W': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-W': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-W': ' ', 'low-R': ' '}

def printBoard(board):
    # Print the Tic-Tac-Toe board
    print(board['top-L'] + '|' + board['top-W'] + '|' + board['top-R'])
    print('---+---+---')
    print(board['mid-L'] + '|' + board['mid-W'] + '|' + board['mid-R'])
    print('---+---+---')
    print(board['low-L'] + '|' + board['low-W'] + '|' + board['low-R'])

turn = 'X'  # Player X starts the game
for i in range(9):
    printBoard(theBoard)  # Print the board before each turn
    print('Tocca a ' + turn + '. In che casella vuoi andare?')  # Ask for the move
    move = input()  # Get the player's move
    theBoard[move] = turn  # Update the board with the player's move
```

```
    # Switch turns between 'X' and 'O'
    if turn == 'X':
        turn = 'O'
    else:
        turn = 'X'

printBoard(theBoard)  # Final board after the game
```

Nested Dictionaries

In Python, you can use nested dictionaries to store and organize complex data. A nested dictionary is a dictionary where the values themselves are dictionaries. Below is an example of how a nested dictionary can be used:

```
allGuests = {'Alice': {'mele': 5, 'pretzel': 12},
             'Bob': {'panini': 3, 'mele': 2},
             'Carol': {'piatti': 3, 'torte di mele': 1}}

def totalBrought(guests, item):
    numBrought = 0
    for k, v in guests.items():
        numBrought = numBrought + v.get(item, 0)
    return numBrought

print('Numero di cose portate:')
print(' - Mele ' + str(totalBrought(allGuests, 'mele')))
print(' - Piatti ' + str(totalBrought(allGuests, 'piatti')))
print(' - Brioches ' + str(totalBrought(allGuests, 'brioches')))
print(' - Panini ' + str(totalBrought(allGuests, 'panini')))
print(' - Torte di mele ' + str(totalBrought(allGuests, 'torte di mele')))
```

String Manipulation in Python

Raw Strings

Raw strings are useful when you want to treat backslashes as literal characters, which is especially helpful for regular expressions or file paths. You can define a raw string by prefixing the string with an `r` or `R`. For example:

```
print(r'Questo - l\'orologio di Marco.')
```

In the example above, the backslash is treated as a literal character.

Multi-line Strings with Triple Quotes

Multi-line strings can be written using triple quotes, either single `'''...'''` or double `"""..."""` quotes. This is useful for strings that span multiple lines. Here's an example:

```
print('''Alice,
the cat of Carlo was arrested for car theft, vagrancy, and extortion.
With all my affection,
Bob''')
```

In this example, the string spans multiple lines while preserving the formatting.

Multi-line Comments

Although Python does not have a specific syntax for multi-line comments, you can use multi-line strings (triple quotes) to create comments. These are technically docstrings, but they can serve the same purpose for adding descriptive comments to your code. Here's an example:

```
"""
This is a test Python program,
written by Al Sweigart at al@inventwithpython.com.
This program is written for Python 3, not Python 2.
"""
def spam():
    """This is a multi-line comment explaining what the spam() function does."""
    print("Hello")
```

Although these triple quotes are intended for docstrings, they can also be used as multi-line comments for clarity.

String Slicing

Python allows you to extract substrings from a string using slicing. You can access specific characters or ranges of characters from a string using the slicing syntax `string[start:end]`.

Here are some examples of string slicing:

```
>>> spam = 'Ciao mondo!'
>>> spam[0]
'C'                  % First character
>>> spam[1]
'o'                  % Second character
>>> spam[-1]
'!'                  % Last character
>>> spam[0:4]
'Ciao'               % Characters from index 0 to 3
>>> spam[:4]
'Ciao'               % Same as above, from the start to index 3
>>> spam[5:]
'mondo!'             % From index 5 to the end
```

Conclusion

This document covered key string manipulation techniques in Python, including raw strings, multi-line strings, and slicing. Mastering these methods will enhance your ability to work with strings efficiently and improve code readability. Experiment with these techniques to strengthen your Python skills.