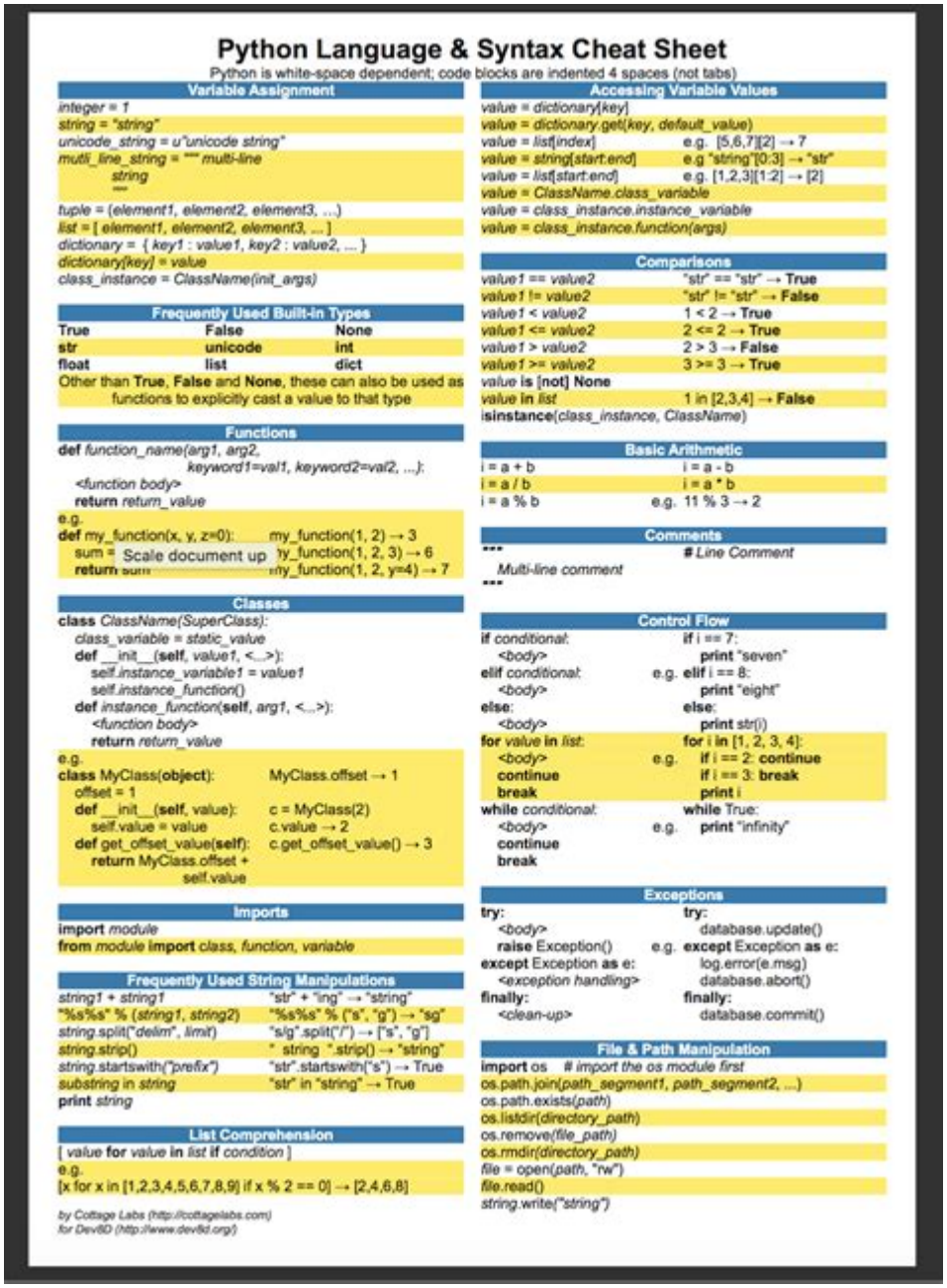
Python Command Cheat Sheet



Python Language & Syntax Cheat Sheet

Python is white-space dependent; code blocks are indented 4 spaces (not tabs)

Variable Assignment

```
integer = 1
string = "string"
unicode_string = u"unicode string"
mutli_line_string = """multi-line
    string
    """
tuple = (element1, element2, element3, ...)
list = [ element1, element2, element3, ... ]
dictionary = { key1 : value1, key2 : value2, ... }
dictionary[key] = value
class_instance = ClassName(init_args)
```

Frequently Used Built-in Types

True	False	None
str	unicode	int
float	list	dict

Other than **True**, **False** and **None**, these can also be used as functions to explicitly cast a value to that type

Functions

```
def function_name(arg1, arg2,
                  keyword1=val1, keyword2=val2, ...):
    <function body>
    return return_value
e.g.
def my_function(x, y, z=0):        my_function(1, 2) → 3
    sum = Scale document up        y_function(1, 2, 3) → 6
    return sum                     my_function(1, 2, y=4) → 7
```

Classes

```
class ClassName(SuperClass):
    class_variable = static_value
    def __init__(self, value1, <...>):
        self.instance_variable1 = value1
        self.instance_function()
    def instance_function(self, arg1, <...>):
        <function body>
        return return_value
e.g.
class MyClass(object):             MyClass.offset → 1
    offset = 1
    def __init__(self, value):     c = MyClass(2)
        self.value = value         c.value → 2
    def get_offset_value(self):    c.get_offset_value() → 3
        return MyClass.offset +
               self.value
```

Imports

```
import module
from module import class, function, variable
```

Frequently Used String Manipulations

string1 + *string1*	"str" + "ing" → "string"
"%s%s" % (*string1*, *string2*)	"%s%s" % ("s", "g") → "sg"
string.split("*delim*", *limit*)	"s/g".split("/") → ["s", "g"]
string.strip()	" string ".strip() → "string"
string.startswith("*prefix*")	"str".startswith("s") → True
substring in *string*	"str" in "string" → True
print *string*	

List Comprehension

```
[ value for value in list if condition ]
e.g.
[x for x in [1,2,3,4,5,6,7,8,9] if x % 2 == 0] → [2,4,6,8]
```

by Cottage Labs (http://cottagelabs.com)
for DevBD (http://www.devbd.org/)

Accessing Variable Values

```
value = dictionary[key]
value = dictionary.get(key, default_value)
value = list[index]                e.g. [5,6,7][2] → 7
value = string[start:end]          e.g "string"[0:3] → "str"
value = list[start:end]            e.g. [1,2,3][1:2] → [2]
value = ClassName.class_variable
value = class_instance.instance_variable
value = class_instance.function(args)
```

Comparisons

value1 == *value2*	"str" == "str" → **True**
value1 != *value2*	"str" != "str" → **False**
value1 < *value2*	1 < 2 → **True**
value1 <= *value2*	2 <= 2 → **True**
value1 > *value2*	2 > 3 → **False**
value1 >= *value2*	3 >= 3 → **True**
value **is [not] None**	
value **in** *list*	1 in [2,3,4] → **False**
isinstance(*class_instance*, *ClassName*)	

Basic Arithmetic

```
i = a + b                          i = a - b
i = a / b                          i = a * b
i = a % b                  e.g. 11 % 3 → 2
```

Comments

```
"""                                # Line Comment
    Multi-line comment
"""
```

Control Flow

```
if conditional:                    if i == 7:
    <body>                             print "seven"
elif conditional:              e.g. elif i == 8:
    <body>                             print "eight"
else:                              else:
    <body>                             print str(i)
for value in list:                 for i in [1, 2, 3, 4]:
    <body>                         e.g.    if i == 2: continue
    continue                               if i == 3: break
    break                                  print i
while conditional:                 while True:
    <body>                         e.g.    print "infinity"
    continue
    break
```

Exceptions

```
try:                               try:
    <body>                             database.update()
    raise Exception()          e.g. except Exception as e:
except Exception as e:                 log.error(e.msg)
    <exception handling>               database.abort()
finally:                           finally:
    <clean-up>                         database.commit()
```

File & Path Manipulation

```
import os    # import the os module first
os.path.join(path_segment1, path_segment2, ...)
os.path.exists(path)
os.listdir(directory_path)
os.remove(file_path)
os.rmdir(directory_path)
file = open(path, "rw")
file.read()
string.write("string")
```

Python command cheat sheet is an invaluable resource for both beginners and experienced developers. It encapsulates essential commands and functions that facilitate Python programming. This article aims to provide a comprehensive overview of frequently used Python commands, categorized for easy reference. Whether you are writing scripts, managing data, or developing applications, this cheat sheet will serve as a handy tool in your programming toolkit.

1. Getting Started with Python

Before diving into commands, it's crucial to set up your Python environment. Here’s a quick guide:

- Install Python from the official website: python.org.
- Choose an Integrated Development Environment (IDE) or code editor. Popular choices include:
    - PyCharm
    - Visual Studio Code
    - Jupyter Notebook
    - Spyder
- Verify your installation by running the command `python --version` or `python3 --version` in your terminal.

2. Basic Python Commands

Understanding the fundamental Python commands is essential for any developer. Below are some basic commands that form the backbone of Python programming.

2.1 Printing and Output

- Print to Console: Use the `print()` function to display output.
```python
print("Hello, World!")
```

- Formatted Strings: Use f-strings (Python 3.6+) for formatted output.
```python
name = "Alice"
print(f"Hello, {name}!")
```

2.2 Variables and Data Types

- Variable Assignment: Assign values using the equals sign.
```python
x = 10
name = "Bob"
```

- Common Data Types:
- Integers: `int`
- Floating-Point Numbers: `float`
- Strings: `str`
- Lists: `list`
- Tuples: `tuple`
- Dictionaries: `dict`

2.3 Basic Arithmetic Operations

Python supports standard arithmetic operations:

- Addition: `a + b`
- Subtraction: `a - b`
- Multiplication: `a b`
- Division: `a / b`
- Integer Division: `a // b`
- Modulus: `a % b`
- Exponentiation: `a b`

3. Control Structures

Control structures are vital for managing the flow of your Python programs.

3.1 Conditional Statements

Use `if`, `elif`, and `else` to control execution flow based on conditions.

```python
if x > 10:
print("x is greater than 10")
elif x == 10:
print("x is equal to 10")
else:
print("x is less than 10")
```

3.2 Loops

Python provides two primary loop constructs: `for` and `while`.

- For Loop: Iterate over a sequence (e.g., list, tuple).
```python
```

```
for i in range(5):
print(i)
```

- While Loop: Repeat as long as a condition is true.
```python
while x < 10:
print(x)
x += 1
```

4. Functions

Functions are reusable pieces of code that perform specific tasks.

4.1 Defining Functions

Use the `def` keyword to define a function.

```python
def greet(name):
return f"Hello, {name}!"
```

4.2 Function Arguments

Functions can take positional and keyword arguments.

```python
def add(a, b=5): b is a default argument
return a + b

print(add(3)) Outputs 8
print(add(3, 4)) Outputs 7
```

5. Data Structures

Python offers various built-in data structures that are crucial for effective programming.

5.1 Lists

Lists are ordered collections that can hold mixed data types.

- Creating a List:
```python
my_list = [1, 2, 3, "four", True]
```

- Accessing Elements:
```python
first_element = my_list[0] 1
```

- List Methods:
- Append: `my_list.append(value)`
- Remove: `my_list.remove(value)`
- Sort: `my_list.sort()`

5.2 Dictionaries

Dictionaries store key-value pairs, offering efficient data retrieval.

- Creating a Dictionary:
```python
my_dict = {"name": "Alice", "age": 25}
```

- Accessing Values:
```python
age = my_dict["age"] 25
```

- Dictionary Methods:
- Add/Update: `my_dict["key"] = value`
- Delete: `del my_dict["key"]`

6. File Handling

Managing files is a common task in Python, allowing you to read from and write to files.

6.1 Reading Files

Use the built-in `open()` function to read files.

```python
with open('file.txt', 'r') as file:
content = file.read()
```

6.2 Writing to Files

You can write to a file using the same `open()` function with write mode.

```python
with open('file.txt', 'w') as file:
file.write("Hello, World!")
```

7. Exception Handling

Handling exceptions is crucial for building robust applications.

```python
try:
x = 10 / 0
except ZeroDivisionError:
print("You cannot divide by zero!")
finally:
print("Execution completed.")
```

8. Libraries and Modules

Python's extensive libraries and modules enhance its functionality.

8.1 Importing Modules

You can import standard or third-party libraries using the `import` statement.

```python
import math
print(math.sqrt(16)) Outputs 4.0
```

8.2 Creating Modules

Create a Python file (e.g., `mymodule.py`) and define functions or classes inside it. Then, you can import this module into your main script.

```python
mymodule.py
def hello():
print("Hello from my module!")

main.py
import mymodule
mymodule.hello() Outputs "Hello from my module!"
```

9. Conclusion

This **Python command cheat sheet** provides a concise overview of essential commands and concepts in Python programming. It serves as a quick reference guide to help you navigate the language's features effectively. Whether you are a novice or an experienced developer, having this cheat sheet at hand will surely enhance your coding efficiency and productivity. Python's versatility, combined with the power of its libraries, makes it a top choice for various applications, from web development to data analysis. Keep this cheat sheet handy as you continue your Python journey!

Frequently Asked Questions

What is a Python command cheat sheet?

A Python command cheat sheet is a concise reference guide that summarizes key Python commands, syntax, and functions, helping developers quickly recall how to perform common programming tasks.

Where can I find a reliable Python command cheat sheet?

You can find reliable Python command cheat sheets on websites like Python.org, GitHub, or educational platforms such as DataCamp and Codecademy, which often provide downloadable PDFs or interactive resources.

What are some essential commands included in a Python cheat sheet?

A Python cheat sheet typically includes essential commands such as data types, control flow statements (if, for, while), functions, list comprehensions, and standard library functions like print(), len(), and range().

How can a Python command cheat sheet improve coding efficiency?

By providing quick access to syntax and commonly used functions, a Python command cheat sheet helps reduce the time spent searching for documentation, thereby increasing coding efficiency and productivity.

Is there a difference between a Python command cheat sheet and a Python tutorial?

Yes, a Python command cheat sheet is a quick reference guide for commands and syntax, while a Python tutorial is a comprehensive instructional resource that teaches concepts, programming techniques, and project-building skills in detail.

Python Command Cheat Sheet

What does colon equal (:=) in Python mean? - Stack Overflow
Mar 21, 2023 · In Python this is simply =. To translate this pseudocode into Python you would need to know the data structures being referenced, and a bit more of the algorithm …

What does asterisk * mean in Python? - Stack Overflow
What does asterisk * mean in Python? [duplicate] Asked 16 years, 7 months ago Modified 1 year, 6 months ago Viewed 319k times

What does the "at" (@) symbol do in Python? - Stack Overflow
Jun 17, 2011 · 96 What does the “at” (@) symbol do in Python? @ symbol is a syntactic sugar python provides to utilize decorator, to paraphrase the question, It's exactly about what does …

Is there a "not equal" operator in Python? - Stack Overflow
Jun 16, 2012 · 1 You can use the != operator to check for inequality. Moreover in Python 2 there was <> operator which used to do the same thing, but it has been deprecated in Python 3.

Using or in if statement (Python) - Stack Overflow
Using or in if statement (Python) [duplicate] Asked 7 years, 6 months ago Modified 8 months ago Viewed 149k times

python - What is the purpose of the -m switch? - Stack Overflow
Python 2.4 adds the command line switch -m to allow modules to be located using the Python module namespace for execution as scripts. The motivating examples were standard library …

What is Python's equivalent of && (logical-and) in an if-statement?
Mar 21, 2010 · There is no bitwise negation in Python (just the bitwise inverse operator ~ - but that

is not equivalent to not). See also 6.6. Unary arithmetic and bitwise/binary operations and …

syntax - What do >> and <
Apr 3, 2014 · 15 The other case involving print >>obj, "Hello World" is the "print chevron" syntax for the print statement in Python 2 (removed in Python 3, replaced by the file argument of the ...

python - Is there a difference between "==" and "is"? - Stack ...
Since is for comparing objects and since in Python 3+ every variable such as string interpret as an object, let's see what happened in above paragraphs. In python there is id function that shows ...

python - What does ** (double star/asterisk) and * (star/asterisk) ...
Aug 31, 2008 · A Python dict, semantically used for keyword argument passing, is arbitrarily ordered. However, in Python 3.6+, keyword arguments are guaranteed to remember insertion ...

What does colon equal (:=) in Python mean? - Stack Overflow
Mar 21, 2023 · In Python this is simply =. To translate this pseudocode into Python you would need to know the data ...

What does asterisk * mean in Python? - Stack Overflow
What does asterisk * mean in Python? [duplicate] Asked 16 years, 7 months ago Modified 1 year, 6 months ago Viewed ...

What does the "at" (@) symbol do in Python? - Stack Overflow
Jun 17, 2011 · 96 What does the “at” (@) symbol do in Python? @ symbol is a syntactic sugar python provides to ...

Is there a "not equal" operator in Python? - Stack Overflow
Jun 16, 2012 · 1 You can use the != operator to check for inequality. Moreover in Python 2 there was <> ...

Using or in if statement (Python) - Stack Overflow
Using or in if statement (Python) [duplicate] Asked 7 years, 6 months ago Modified 8 months ago Viewed 149k ...

Unlock your Python potential with our ultimate Python command cheat sheet! Discover essential commands and tips. Learn more to enhance your coding skills today!

Back to Home