Python Programming Language Cheatsheet

Updated on 20 January 2024 by Huzaif Sayyed

Python, a dynamically-typed language known for its readability and versatility. Dive into Python with this Cheatsheet for a quick start on its fundamentals.

Links

Python Official Website

★ Want More Cheat Sheet

Download PDF

Hello World! Python Program

Create a file called hello.py

```
print("Hello World")
```

Use `python hello.py` to build and run it. In this example will give output: Hello, World!

Basics

Comments

```
# This is a single-line comment

'''
This is a
multi-line
comment
'''
```

For single line use `#` and for multi line comments use `'''`

Variables

```
x = 5
y = "Hello, Python!"
```

Python variables are dynamically typed, meaning you don't need to declare their type explicitly, and they can change type during runtime.

Data Types

```
int_num = 10
float_num = 3.14
string = "text"
boolean = True
```

Print

```
print("Hello!")
```

This will print "Hello!" text in console.

Input

```
user_input = input("Enter something: ")
```

Once you run the code, it will ask for input and store in variable.

Type Conversion

```
str_to_int = int("5")
float_to_int = int(3.14)
int_to_str = str(10)
```

Arithmetic Operators

```
result = 5 + 3
result = 10 - 3
result = 2 * 4
result = 9 / 3
result = 7 % 2
result = 2 ** 3
```

Comparison Operators

```
x == y   # Equal to
x != y   # Not equal to
x > y    # Greater than
x < y    # Less than
x >= y   # Greater than or equal to
x <= y   # Less than or equal to
```

Control Flow

if-else statement

```
if condition:
    # code to execute if condition is True
else:
    # code to execute if condition is False
```

elif statement

```
if condition1:
    # code to execute if condition1 is True
elif condition2:
    # code to execute if condition2 is True
else:
    # code to execute if all conditions are False
```

for loop

```
for item in iterable:
    # code to execute for each item in the iterable
```

while loop

```
while condition:
    # code to execute while condition is true
```

break and continue

```
for i in range(5):
    if i == 2:
        break  # exit the loop
    if i == 1:
        continue  # skip the rest of the loop for this
    print(i)
```

Data Structures

Lists

```
my_list = [1, 2, 3, "four", 5.0]
```

Lists are mutable arrays

Tuple

```
my_tuple = (1, 2, 3)
```

Tuples are immutable.

Set

```
my_set = {1, 2, 3}
```

Sets store unique elements.

Dictionary

```
my_dict = {'key': 'value', 'name': 'John'}
```

Dictionaries use key-value pairs.

Accessing elements

```
element = my_list[0]
element = my_tuple[1]
element = my_dict['key']
```

Updating elements

```
my_list[0] = 10
my_tuple = (4, 5, 6)
my_set.add(4)
my_dict['age'] = 25
```

Removing elements

```
my_list.remove(2)
my_set.discard(3)
del my_dict['key']
```

Common operations on sequences

```
len(my_list)
max(my_tuple)
min(my_set)
```

Functions

Defining a function

```
def greet(name):
    return "Hello, " + name + "!"
```

Calling a function

```
result = greet("Alice")
```

Default parameter

```
def greet(name, greeting="Hello"):
    return greeting + ", " + name + "!"
```

Variable number of arguments

```
def add(*args):
    return sum(args)
```

Lambda function

```
square = lambda x: x ** 2
```

File Handling

Reading from a file

```
with open("filename.txt", "r") as file:
    content = file.read()
```

Writing to a file

```
with open("new_file.txt", "w") as file:
    file.write("Hello, Python!")
```

Appending to a file

```
with open("existing_file.txt", "a") as file:
    file.write("\nAdding a new line.")
```

Exception Handling

```
try:
    # code that may raise an exception
except SomeException as e:
    # handle the exception
else:
    # code to execute if no exception is raised
finally:
    # code to execute regardless of whether an exception is raised or not
```

Decorators

Decorators in Python are functions that modify or extend the behavior of other functions. They provide a concise way to enhance or wrap functions without modifying their source code.

Simple decorator

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the funct
        func()
        print("Something is happening after the functi
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")
```

Decorator with arguments

```
def repeat(n):
    def decorator(func):
        def wrapper(*args, **kwargs):
            for _ in range(n):
                result = func(*args, **kwargs)
            return result
        return wrapper
    return decorator

@repeat(3)
def greet(name):
    print(f"Hello, {name}!")
```

Advanced Data Structures

Collections module

```
from collections import Counter, defaultdict, OrderedDi
```

Counter

```
my_list = [1, 2, 2, 3, 3, 3, 4]
counter = Counter(my_list)  # Count occurrences
```

defaultdict

```
my_dict = defaultdict(int)
```

Default value for nonexistent keys

OrderedDict

```
ordered_dict = OrderedDict()
ordered_dict['a'] = 1
ordered_dict['b'] = 2
```

Preserves the order of insertion

List Comprehensions

List comprehensions in Python provide a concise way to create lists. They allow you to generate a new list by applying an expression to each item in an existing iterable.

```
squares = [x**2 for x in range(5)]
```

Conditional list comprehension

```
even_squares = [x**2 for x in range(10) if x % 2 == 0]
```

Object-Oriented Programming (OOP)

Class and object

```
class MyClass:
    def __init__(self, attribute):
        self.attribute = attribute

    def method(self):
        print("Method called")
```

Creating an object

```
obj = MyClass("value")
```

Inheritance

```
class ChildClass(MyClass):
    def __init__(self, attribute, new_attribute):
        super().__init__(attribute)
        self.new_attribute = new_attribute
```

Modules and Packages

```
# Importing modules
import math
from math import sqrt
from mymodule import my_function

# Creating a module (mymodule.py)
def my_function():
    print("Module function")

# Creating a package (mypackage/__init__.py)
# Package can contain multiple modules
```

Modules are Python files containing code, and packages are directories that contain multiple modules.

Congratulations on reaching the end of this Python Programming Language Cheatsheet! This resource is designed to make your coding experience easy and efficient. Feel free to bookmark this page or download the PDF for future reference. Happy Zig Programming!