Learn to Code - Python Cheat Sheet

Comments

```
# This is a comment. Comments start with a # character.
# Comments are ignored by the interpreter.
# Use comments to make notes in your program!
```

Input / Output

```
print("This will be output to the terminal.")
answer = input("This is a question for the user. Their input will
be stored in the variable 'answer'.")
# The program pauses at the 'input' command to wait
# for the user's input, before continuing.
```

Basic Data Types

Integers - Integer numbers.
Floating Points ('floats') - Numbers with decimal points.
Strings - Text.
Booleans - True or False.

```
int(some_variable)    # Convert to integer
float(some_variable)  # Convert to float
str(some_variable)    # Convert to string
```

Variables

```
my_string_variable = "This is a string!"
my_integer_variable = 42
message = "We can output variables with print."
print(message)
```

Numeric Expressions

```
x + y           # Addition
x - y           # Subtraction
x * y           # Multiplication
x / y           # Division
x ** y          # Power
x % y           # Modulo
(x + y) * z     # Brackets
```

String Expressions

```
"Hello" + " World!"        # Concatenation
"badger" * 3               # Repetition

# String formatting:
# %s - String
# %d - Integer
```

```
# %f - Float
# %e - Float in scientific notation
"My name is %s and I am %d years old." % ("Alex", 14)
```

Boolean Expressions

```
# Basic boolean values - note capitalisation!
True
False

# Comparison Operators
# These take two variables of any time, and evaluate to a boolean
x == y          # Equality - do not confuse with assignment!
x != y          # Inequality
x < y           # Less than
x > y           # Greater than
x <= y          # Less than or equal to
x >= y          # Greater than or equal to

# Logical Operators
# These take one or two booleans, and evaluate to a boolean
not x           # Inverts value
x and y         # Only True if x and y are both True
x or y          # True if either x or y (or both) are True
```

If / Elif / Else

```
if some_boolean_expression:
    # Do some stuff if the expression is True
elif some_other_boolean_expression:
    # If the first expression is False, but
    # this expression is True, do this
else:
    # If none of the above, do this
```

While Loops

```
while some_boolean_expression:
    # Do some stuff repeatedly, until the
    # expressions is False
```

Lists

```
my_string_list = ["red", "green", "blue"]
my_integer_list = [4, 8, 3, 10]
my_mixed_list = ["red", 12, 4, True, "green"]
```

List Operations

```
my_list = ["a", "b", "c", "d", "e", "f"]
my_list[0]           # This is the first element - "a"
```

```
my_list[2]            # This is the third element - "c"
my_list[2:4]          # List containing elements two and three
                      # i.e. the list ["c", "d"]
my_list + ["g", "h"]  # Concatenation
                      # ["a", "b", "c", "d", "e", "f", "g", "h"]
["cat", "dog"] * 3    # Repetition
                      # ["cat", "dog", "cat", "dog", "cat", "dog"]
len(my_list)          # Length of list - 6
"c" in my_list        # Is the value in the list - True
"z" in my_list        # False
x, y = [6, 3]         # List decomposition - now x=6 and y=3
```

For Loops

```
for x in my_list:
    # Run this code for each item in the list in turn
    # The current item is stored in x
    print(x)          # So this will output each value in the list

range(10)             # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
range(4, 10)          # [4, 5, 6, 7, 8, 9]

# This will loop over the values 0-9 and output them:
for i in range(10):
    print(i)
```

Sets

```
# Sets cannot contain duplicates, and the values are not stored
# in any particular order
# List operations work as normal on sets
my_set = {1, 3, 5, 7}
```

Tuples

```
# Tuples cannot be modified once created
# List operations work as normal on tuples, except those that
# would modify the tuple
my_tuple = (6, 3)
```

Dictionaries

```
# Dictionaries store key-value pairs
my_dict = {"Alice": 26, "Bob": 60, "Carol": 30, "Dave": 16}
# Access values using the key
print(my_dict["Bob"])      # Outputs 60
# To use a for loop with a dictionary:
for (key, value) in my_dict.items():
    # Do stuff
```

Functions

```
# A function without parameters or return value
def say_hi():
    print("Hello World!")

# A function with parameters, including a default parameter
def greet(greeting, name = "User"):
    print("%s, %s!" % (greeting, name))

# A function that returns a value
def add(x, y):
    return x + y

# Functions can return early
def early_terminate():
    print("The function will terminate now")
    return
    print("This message will never be displayed")

# A function taking a variable number of parameters
def flexi_func(x, *params):
    # x is a regular parameter
    print(x)
    # params is a tuple containing all remaining parameters
    for y in params:
        print(y)

# Calling a function
say_hi()                    # No parameters
greet("Hello", "Alex")      # Parameters
greet("Hi there")           # Using a default parameter
total = add(3, 6)           # Storing returned value in a variable
flexi_func("test")          # "test" will be the x parameter
                            # params is empty in this case
flexi_func("test", 4, 2, 7, 2, 3)     # But we could have many
                                      # more parameters!

# Functions can be recursive
def factorial(x):
    if x <= 1:
        return 1
    else:
        return x * factorial(x - 1)

print(factorial(5))         # 120
```

Modules

```
# Bring in the math module
import math

# We can then use functions and variables of the module
y = math.sin(0.5)
rad_ang = math.radians(60)

# We can import specific parts of a module directly into the
# main namespace
from math import sin, cos, tan

a = sin(0.5)
b = cos(0.5)
c = tan(0.5)

# We could bring in the whole module directly into the main
# namespace, but this is usually unwise
from math import *
```

A list of Python modules can be found at:
https://docs.python.org/3/py-modindex.html

Classes and Objects

```
# A class representing the concept of a rectangle
class Rectangle:
    # Initialiser method - called automatically when
    # a new rectangle is made
    def __init__(self, w, h):
        # Save the passed-in parameters as member variables
        self.width = w
        self.height = h

    # Calculate perimeter of this rectangle
    def perimeter(self):
        return 2 * (self.width + self.height)

    # Calculate area of this rectangle
    def area(self):
        return self.width * self.height

# Create some rectangle objects, using our rectangle class
small_rect = Rectangle(1.5, 2.0)
big_rect = Rectangle(80.2, 45.6)
```

```
# Output the perimeter of the big rectangle
print(big_rect.perimeter())

# Output the area of the small rectangle
print(small_rect.area())

# A class representing an animal
class Animal:
    # Initialiser method
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.food_eaten = 0

    # Eat some food
    def eat(self, amount):
        self.food_eaten += amount

    # Make a generic animal noise
    def speak(self):
        print("%s makes a noise!" % (self.name))

# A class representing a dog, derived from animal
class Dog(Animal):
    # Initialiser method - replaces parent class initialiser
    def __init__(self, name, age, breed):
        # Call the parent's initialiser first
        Animal.__init__(self, name, age)
        # Now initialiser our own members
        self.breed = breed

    # Make a dog noise - replaces animal's generic noise
    def speak(self):
        print("%s goes woof!" % (self.name))

    # The dog also has access to the `eat` method and all the
    # member variables of animal, because it inherits these
    # from the parent class
```

File Access

```
# Open a file for writing, write some information, and close
file1 = open("logfile.txt", "w")
file1.write("A line of information.\n")
file1.write("This bit ")
file1.write("is on the same line as this bit.")
file1.close()
```

```
# Open a file for reading, read it in different ways, and close.
file2 = open("example.txt", "r")
data1 = file2.read(4)           # Read first 4 bytes
data2 = file2.readline()        # Read the rest of the line
data3 = file2.readline()        # Read the next line
data4 = file2.read()            # Read the entire rest of the file
file2.close()

# Open the same file a second time
file3 = open("example.txt", "r")
# Read each line of the file into a separate string in a list
datalines = file3.readlines()
# Close the file
file3.close()
```