Python basics

2022

Paul Templier

Finding answers

Documentation

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Error messages

```
print(a)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

```
print("hello"())
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object is not callable
```

Variables

Variable types

```
a = 3 # int: Integer
b = 1.23 # float: Floating point value

c = "d" # string
d = "hello" # string

e = ["a", "b", "c"] # list
f = {"a":23, "b": 34} # dict

print(x) # Display the value of variable x
type(x) # Get the type of a variable x

g = None # Nothing, has no type
```

Strings

```
# Definition
a = "" # Empty
a = "hello world"

# Access (like a list, index starts at 0)
b = a[4] # "o"

# Length
len(a) # 11
```

Useful string functions

```
a = "hello world"
print(a)

a.count("l") # 3

a.replace("l", "L") # Returns "heLLo worLd"

a.split() # ["hello", "world"]
a.split("l") # ["he", "", "o wor", "d"]

"/".join(["hello", "world"]) # "hello/world"
```

Data structures

Lists

```
# Definition
l = [] # Empty
l = [1, "hello"] # Not empty

len(l) # Length

# Access: index starts at 0
l = ["Alice", "Bob", "Charles"]
b = l[2] # "Charles"

l[1] = "Bernard" # l = ["Alice", "Bernard", "Charles"]
l.append("David")  # ["Alice", "Bernard", "Charles", "David"]
```

Dictionaries: {key:value}

```
# Definition
d = {} # Empty
d = {"Alice": 12, "Bob": 35} # Key: value

# Access with key
a = d["Bob"] # 35

# Edition
d["Bob"] = 32 # {"Alice": 12, "Bob": 32}

# Addition
d["Charles"] = 45 # {"Alice": 12, "Bob": 32, "Charles":45}
```

Functions

Reusing blocks of code

Functions

```
# Definition
def f():
    print("Hello world!")

# Use
f() # Hello world!

# Return a value
def f():
    return 3

a = f() # a = 3
```

Function arguments

```
# Definition
def f(a, b):
    return a * b

# Use
x = f(3, 4) # x = 12

# Default value
def f(a=1):
    return a * 3

x = f() # x = f(1) = 3
x = f(4) # x = 12
```

Variable scope

```
a = 1 # defined everywhere in this code

def f(x, y): # x and y defined inside f only
    z = 4 # z defined inside f only
    print(x, y)
    print(a)

f(2, 3)
# 2 3
# 1

print(x, z) # Undefined: only exist inside f
```

Conditions and loops

If condition

```
def f(x):
    if x < 18:
        print("Kid")
    elif x > 90: # Optional
        print("Old")
    else: # Optional, happens when the condition is not verifies
        print("Adult")

f(12) # Kid
f(35) # Adult
f(123) # Old
```

For loop

```
l = [23, 45, 67, 12]
for i in l:
    print(i) # 23, 45, 67, 12
```

```
for i in range(5):
    print(i) # 0, 1, 2, 3, 4
```

Iterating on a dictionary

```
d = {"Alice": 12, "Bob": 35, "Charles": 42} # Key: value

# Iterate on the keys
for k in d.keys():

#Iterate on values
for v in d.values():

# Iterate on both
for k, v in d.items():
```

While loop

```
a = 1
while a < 34:
    a = a * 2

print(a) # 64

while True: # Infinite loop
    print("One more time")

# Keywords
pass # Do nothing
continue # Get to the next iteration
break # Stop the loop
```

Object Oriented Programming

Object Oriented Programming

Class:

Blueprint of a concept (*eg* the concept of a car)

Object:

Instance of a class (*eg my* car, Bob's car)

Classes

```python
class Car: # Definition
    def __init__(self, nb_wheels, owner): # Method called when creating an object
        self.wheels = nb_wheels
        self.owner = owner

    def drive(self, distance):
        print(f"Vrooom: {self.owner}'s car is driving for {distance} km")

my_car = Car(4, "Paul") # my_car is an object
bob_car = Car(3, "Bob") # bob_car is a different object

print(my_car.wheels) # 4
my_car.wheels = 3 # Set value

my_car.drive(3) # Vrooom: Paul's car is driving for 3 km"
```

Useful class methods

```
class TestClass:
    def __str__(self):
        # Method called when printing an object, return a string
        return "Test object"

    def __len__(self):
        # Method called when calling len(x)
        return 1

    def __getitem__(self, key):
        # Method called when using x[key]
        return None
```

Inheritance

```python
class Vehicle: # Parent class
    def __init__(self, nb_wheels, owner): # Method called when creating an object
        self.wheels = nb_wheels
        self.owner = owner

    def drive(self, distance):
        print(f"Vrooom: {self.owner}'s vehicle is driving for {distance} km")

class Car(Vehicle): # Child class inheriting from parent
    def __init__(self, nb_wheels, owner, brand):
        super(self).__init__(nb_wheels, owner) # Setup parent class parameters
        self.brand = brand # Additional field

    def speed(self): # Add method
        print("Car goes Vroooooom")
```

Misc

Import

```python
import math # Import a module
a = math.sqrt(3) # Get an element in the module

from math import sqrt, floor # Import specific elements
a = sqrt(3) # The element is already available

from math import * # Import all the elements from math
a = sqrt(3)
b = floor(2.34)

import math.sqrt as racine # Rename the module or element
a = racine(3)
```