- 6 Reasons Why Python Is Suddenly Super Popular
- Surprising Facts Why Python is gaining more popularity among Developers

Contents x

Basics

Hello World code

```
print("Hello World")
# prints "Hello World"
# above line is a comment
'''
This is a multi-line
comment in python
'''
```

Data types code

```
# declare an integer
a = 12
print a       # prints "12"
print type(a) # prints <type 'int'>

# declare a float
b = 1.7
print b       # prints "1.7"
print type(b) # prints <type 'float'>

# declare a string
c = "Python"
print c       # prints "Python"
print type(c) # prints <type 'str'>

# declare a boolean
d = True
print d       # prints "True"
print type(d) # prints <type 'bool'>
```

Multiple variable assignments code

```
# assign values to multiple variables in a single line
a, b, c = 1, 2, 3
```

```
a, b, c = 1, 2, 3

# assign values with different data types to multiple variables in a single line
a, b, c = 1, 3.5, "hello"

print type(a) # prints <type 'int'>
print type(b) # prints <type 'float'>
print type(c) # prints <type 'str'>
```

Contents x

Math operations

code

```
a = 10

print a + 1  # Addition: prints "11"
print a - 1  # Subtraction: prints "9"
print a * 2  # Multiplication: prints "20"
print a / 2  # Division: prints "5"
print a  2 # Exponentiation: prints "100"
```

Logical operations

code

```
# declare some boolean variables
x = True
y = False

print x and y  # prints "False"
print x or y   # prints "True"
print not x    # prints "False"
print x & y    # prints "False"
```

Conditions

code

```

# if-elif-else statements

import random
a = [50, 100, 200, 300]

# pick a random number from the list "a"
b = random.choice(a)

# the conditionals
if (b < 100):
    print("Number - " + str(a) + " is less than 100")
elif (b >= 100 and b < 200):
```

```
    print("Number - " + str(b) + " is greater than or equal to 100 but less than 20
else:
    print("Number -" + str(b) + " is greater than or equal to 200")

# for me it prints
# Number - 100 is greater than or equal to 100 but less than 200
```

Contents x

Loops

code

```
# while loop
c = 0
while (c < 10):
    print (c,  end='')
    c += 1

# prints 0123456789

# for loop
numbers = [1, 2, 4]
for x in numbers:
    print x

# prints 1
#        2
#        4

x = 5
for c in range(x):
    print (c)

# prints 0
#        1
#        2
#        3
#        4
```

Strings

code

```

# declare two strings
a = "Python"
b = " is awesome!"

print len(a)                    # Length of the string: prints "6"
print len(b)                    # prints "12"
print a + b                     # String concatenation: prints "Python is awesc
print a, b                      # prints "Python  is awesome!"
print "{}{}".format(a, b)       # prints "Python is awesome!"
print "%s%s" % (a, b)           # sprintf style formatting: prints "Python is a
```

```
print a.upper()                    # converts all characters to uppercase: prints
print a.lower()                    # converts all characters to lowercase: prints
print b.strip()                    # removes trailing and leading whitespaces: pri
print b.replace("awesome", "great") # replace a substring with a new str
```

Contents x

Lists

code

```

# declare a list
l = [1,2,3,4,5]

# length of list
print len(l)      # prints "5"

# indexing
print l[0]        # prints "1"
print l[1]        # prints "2"
print l[len(l)-1] # prints "5"
print l[-1]       # negative indexing: prints "5"

# insert and remove
l.append(6)        # inserts "6" at last
print l            # prints "[1,2,3,4,5,6]"
item = l.pop()     # removes last element and returns that element
print item         # prints "6"
l.append("string") # adds different data type too
print l            # prints "[1,2,3,4,5,'string']"
l.pop()            # removes last string element

# slicing list
print l[1:2]       # prints "2"
print l[1:3]       # prints "2,3"
print l[0:]        # prints "[1,2,3,4,5,'string']"
print l[0:-1]      # prints "[1,2,3,4,5]"
print l[:]         # prints "[1,2,3,4,5,'string']"

# loop over the list
for item in l:
    print item     # prints each item in list one by one

# enumerate over the list
for i, item in enumerate(l):
    print "{}-{}".format(i, item) # prints each item with its index


# squaring elements in a list
for item in l:
    if item%2 == 0:
        print item2      # square each even number in the list

# above can be achieved using a list comprehension too! (one-line)
print [x2 for x in l if x%2==0]

# sort the list

h = [5, 7, 2, 4, 9]
```

```
49
50  # ascending order
51  b.sort()
52  print b  # prints [2, 4, 5, 7, 9]
53
54  # descending order
55  b.sort(reverse=True)
56  print b # prints [9, 7, 5, 4, 2]
57
58  # reverse the list (notice this is not descending order sort)
59  a = ["dhoni", "sachin", "warner", "abd"]
60  a.reverse()
61  print a # prints ['abd', 'warner', 'sachin', 'dhoni']
62
63  # count of object in list
64  a = [66, 55, 44, 22, 11, 55, 22]
    print a.count(22) # prints 2
```

Contents x

Tuples

code

```
1   # declare a tuple
2   t = (500, 200)
3
4   print type(t)    # prints "<type 'tuple'>"
5   print t[1]       # prints 200
6
7   # tuple of tuples
8   tt = ((200,100), t)
9
10  print tt         # prints "((200, 100), (500, 200))"
11  print tt[1]      # prints "(500, 200)"
12
13  # loop over tuple
14  for item in t:
15      print item   # prints each item in the tuple
16
17
18  # built-in tuple commands
19  print len(t) # prints the length of tuple which is 2
20  print max(t) # prints the max-valued element which is 500
21  print min(t) # prints the min-valued element which is 200
22
23  # convert list to tuple
24  l = [400, 800, 1200]
25  l_to_t = tuple(l)
26
27  print type(l_to_t) # prints <class 'tuple'>
```

Set

code

```
# set is a collection of unordered and unindexed data which is written with curly b
s = {"ironman", "hulk", "thor", "thanos"}

for x in s:
    print(x)

'''
prints
ironman
thor
hulk
thanos
'''

# check if value exist in set
if "thanos" in s:
    print("endgame") # prints 'endgame'

# add a single item to a set using 'add'
s.add("rocket")

# add multiple items to a set using 'update'
s.update(["blackhood", "blackwidow"])

# get length of a set
print(len(s)) # prints 7

# 'remove' or 'discard' an item from the set
# 'remove' raise an error if item to remove does not exist
# 'discard' will not raise any error if item to remove does not exist
s.remove("thanos")
s.discard("blackwidow")

# clear the set
s.clear()

# delete the set
del s
```

Contents x

Dictionaries

code

```
# declare a dictionary
d = { "1" : "Ironman",
      "2" : "Captain America",
      "3" : "Thor"
    }

print type(d)    # prints "<type 'dict'>"
print d["1"]     # prints "Ironman"

# loop over dictionary
for key in d:
    print key    # prints each key in d
```

```
    print d[key] # prints value of each key in d (unsorted)

# change values in the dictionary
d["2"] = "Hulk"
for key, value in d.items():
    print(key + " - " + value)

'''
prints
1 - Ironman
2 - Hulk
3 - Thor
'''

# check if key exists in a dictionary
if "3" in d:
    print("Yes! 3 is " + d["3"])
# prints 'Yes! 3 is Thor'

# get length of the dictionary
print(len(d)) # prints 3

# insert a key-value pair to a dictionary
d["4"] = "Thanos"

# remove a key-value pair from the dictionary
d.pop("4")

# same thing using 'del' keyword
del d["2"]

# clear a dictionary
d.clear()
```

Contents x

Exception Handling

code

```

# try-except-finally
# try: test a block of code for errors.
# except: allows handling of errors.
# finally: execute code, regardless of the result of try and except blocks.
try:
    print(x)
except:
    print("Something is wrong!")

# prints 'Something is wrong!' as x is not defined

try:
    print(x)
except:
    print("Something is wrong!")
finally:
    print("Finally always execute after try-except.")


# prints
```

```
# prints
# Something is wrong!
# Finally always execute after try-except.
```

Contents x

Functions

code

```
def squared(x):
    return x*x

print squared(2)   # prints "4"
```

Intermediate

Lambda

code

```
# a lambda function = a small anonymous function
# takes any number of arguments, but can have only one expression
# lambda arguments : expression

# lambda function with one argument
add_hundred = lambda x : x + 100
print(add_hundred(5)) # prints 105

# lambda function with multiple arguments
multiply = lambda a, b, c : a*b*c
print(multiply(10,5,10)) # prints 500
```

List Comprehensions

code

```

nums = [1, 2, 3, 4, 5, 6, 7]

# traditional for loop
l = []
for n in nums:
    l.append(n)
print(l) # prints '[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]'

# meet list comprehension
l = [n for n in nums]
print(l) # prints '[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]'

# get square of each number
```

```
# get square of each number
l = [n*n for n in nums]
print(l) # prints '[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]'

# same thing achieved using 'map' + 'lambda'
# 'map' means running everything in the list for a certain function
# 'lambda' means an anonymous function
l = map(lambda n: n*n, nums)
for x in l:
    print(x)

# prints
# 1
# 4
# 9
# 16
# 25
# 36
# 49
# 64
# 81
# 100

# using 'if' in list comprehension
l = [n for n in nums if n%2 == 0]
print(l) # prints '[2, 4, 6, 8, 10]'

# returning tuples with two for loops in list comprehension
l = []
for letter in "ab":
    for num in range(2):
        l.append((letter, num))
print(l) # prints '[('a', 0), ('a', 1), ('b', 0), ('b', 1)]'

# same thing using list comp
l = [(letter, num) for letter in "ab" for num in range(2)]
print(l) # prints '[('a', 0), ('a', 1), ('b', 0), ('b', 1)]'
```

Contents x

Dict Comprehensions

code

```
names = ["Robert Downey Jr", "Chris Evans", "Chris Hemsworth", "Mark Ruffalo"]
heros = ["Ironman", "Captain America", "Thor", "Hulk"]

# traditional dictionary using zip()
d = {}
for name, hero in zip(names, heros):
    d[name] = hero
for name in d:
    print(name + " - " + d[name])

'''
prints
Mark Ruffalo - Hulk
Chris Hemsworth - Thor
Robert Downey Jr - Ironman
Chris Evans - Captain America
```

```
Chris Evans - Captain America
'''

# meet dict comprehension
d = {name: hero for name, hero in zip(names, heros)}
for name in d:
    print(name + " - " + d[name])

'''
prints
Mark Ruffalo - Hulk
Chris Hemsworth - Thor
Robert Downey Jr - Ironman
Chris Evans - Captain America
'''

# dict comprehension with condition
d = {name: hero for name, hero in zip(names, heros) if name != "Mark Ruffalo"}
for name in d:
    print(name + " - " + d[name])

'''
prints
Chris Hemsworth - Thor
Robert Downey Jr - Ironman
Chris Evans - Captain America
'''
```

Contents x

Set Comprehensions

code

```
nums = [1, 1, 2, 1, 3, 4, 4, 5, 5, 6, 7, 8, 8, 9]

# traditional set (list of unique elements)
s = set()
for n in nums:
    s.add(n)
print(s) # prints {1, 2, 3, 4, 5, 6, 7, 8, 9}

# meet set comprehension
s = {n for n in nums}
print(s) # prints {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Generator Expressions

code

```

# I need to yield 'n*n' for each 'n' in nums
nums = [1,2,3,4,5,6,7,8,9]

# traditional generator function
def gen_func(nums):
    for n in nums:
```

```
        yield n*n

m = gen_func(nums)
for i in m:
    print(i)

# generator expression
m = (n*n for n in nums)
for i in m:
    print(i)

'''
both prints
1
4
9
16
25
36
49
64
81
'''
```

Contents x

Modules

Regular Expressions

re rules

```
import re

# multi-line string example
str = '''
Rahul is 19 years old, and Ashok is 24 years old.
Murali is 65, and his grandfather, Karthik, is 77.
'''

# findall()
ages  = re.findall(r'\d{1,3}', str)
names = re.findall(r'[A-Z][a-z]*', str)
print ages  # prints ['19', '24', '65', '77']
print names # prints ['Rahul', 'Ashok', 'Murali', 'Karthik']

# finditer()
ages = re.finditer(r'\d{1,3}', str)
for m in ages:
    print(m.group())

# prints
# 19
# 24
# 65
# 77
```

```
# split()
str = "This is an example string"
splitted = re.split(r'\s*', str)
print splitted # prints ['This', 'is', 'an', 'example', 'string']

# match()
str = "Dogs are braver than Cats"
matches = re.match(r'[A-Z][a-z]*', str)
print matches.group() # prints "Dogs"

# search()
str = "For data science help, reach support@datacamp.com"
searches = re.search(r'([\w\.-]+)@([\w\.-]+)', str)
print searches.group()  # prints support@datacamp.com
print searches.group(1) # prints support
print searches.group(2) # prints datacamp.com
```

Contents x

os module

code

```

# os module is a powerful module in python
import os

# get current working directory
print(os.getcwd()) # prints 'G:\\workspace\\Python'

# change current working directory
os.chdir("G:\\workspace\\python\\learning")
print(os.getcwd()) # prints 'G:\\workspace\\python\\learning'

# list directories in the current working directory
print(os.listdir()) # prints ['built-ins', 'Lists', 'numpy', 'strings']

# create a directory in the current working directory
os.mkdir("dicts")
os.makedirs("dicts/nested-dicts")

# remove a directory in the current working directory
os.rmdir("dicts")
os.removedirs("dicts/nested-dicts")

# rename a file or directory
os.rename("Lists", "lists")

# stats of a file or directory
os.stat("lists")
# prints 'os.stat_result(st_mode=16895, st_ino=281474977861215, st_dev=4143122855,

# traverse a directory tree
for dirpath, dirnames, filenames in os.walk("G:\\workspace\\python\\learning"):
    print("Current Path: ", dirpath)
    print("Directories: ", dirnames)
    print("Files: ", filenames)
    print()

```

```
'''
prints
Current Path:  G:\workspace\python\learning
Directories:  ['Built-ins', 'lists', 'NumPy', 'Strings']
Files:  []

Current Path:  G:\workspace\python\learning\Built-ins
Directories:  []
Files:  ['evalu.py', 'input.py', 'zipped.py']

Current Path:  G:\workspace\python\learning\lists
Directories:  []
Files:  ['list_01.py', 'tuple_01.py']

Current Path:  G:\workspace\python\learning\NumPy
Directories:  []
Files:  ['ceilr.py', 'concatenate.py', 'eye_identity.py', 'flatten.py', 'math.py',

Current Path:  G:\workspace\python\learning\Strings
Directories:  []
Files:  ['formatting.py']
'''

# check if a file exist
print(os.path.isfile("G:\\workspace\\python\\learning\\Strings\\formatting.py"))
# prints 'True'

# check if a directory exist
print(os.path.exists("G:\\workspace\\python\\learning\\Strings"))
print(os.path.isdir("G:\\workspace\\python\\learning\\Strings"))
# both prints 'True'

# accessing environment variable
print(os.environ.get("HOME"))
# prints 'C:\\Users\\Gogul Ilango'
```

Contents x

sys module

code

```

# sys module is used to parse input arguments given to a python file.
# this is used if you call a python script with arguments in command line.
import sys

firstarg  = ""
secondarg = ""
try:
    firstarg  = sys.argv[1]
    secondarg = sys.argv[2]
except:
    if (firstarg == ""):
        print("No first argument!")
    if (secondarg == ""):
        print("No second argument!")


# error text
```

```
sys.stderr.write("This is stderr text\n")
sys.stderr.flush()
sys.stdout.write("This is stdout text\n")
```

Contents x

OOP

Classes

code

```
# create a class
class Customer(object):

    # init method is a must
    def __init__(self, name, age):
        self.name = name
        self.age  = age

    # a simple print method
    def print_customer(self):
        print ("Customer: {}, Age: {}".format(self.name, self.age))

# define an instance
a = Customer("Gogul", "24")
a.print_customer() # prints "Customer: Gogul, Age: 24"
```

Class Variables

code

```
class Customer(object):

    # this is a class variable
    raise_amount = 1.05
    num_of_custs = 0

    def __init__(self, name, age, pay):
        self.name = name
        self.age  = age
        self.pay  = pay

        Customer.num_of_custs += 1

    def apply_raise(self):
        print ("Customer {} new pay is {}".format(self.name, float(self.pay) * self

if __name__ == '__main__':
    # class variable not updated
    a = Customer("Gogul", "24", "5000")
    a.apply_raise() # prints Customer Gogul new pay is 5250.0
```

```
22
23      # class variable updated
24      b = Customer("Mahadevan", "25", "6000")
25      b.raise_amount = 2.05
26      b.apply_raise() # Customer Mahadevan new pay is 12299.999999999998
27
28      # print dict of an instance
29      print (a.__dict__) # prints {'name': 'Gogul', 'age': '24', 'pay': '5000'}
30
31      # There are 2 customers
        print ("There are {} customers".format(Customer.num_of_custs))
```

Contents x

How to's

How to handle files? code

```
1   '''
2   four different methods (modes) to open a file
3   "r" - read; default mode; opens a file for reading, error if the file does not exis
4   "a" - append; opens a file for appending, creates the file if it does not exist.
5   "w" - write; opens a file for writing, creates the file if it does not exist.
6   "x" - create; creates the file, returns an error if the file exist.
7
8   "t" - text; default/text mode
9   "b" - binary; binary mode
10  '''
```

How to read file line-by-line? code

```
1
2   # not the memory efficient way
3   filename = "entry_1.txt"
4   with open(filename) as f:
5       data = f.readlines()
6   # remove whitespaces at end of each line
7   data = [x.strip() for x in data]
8
9   # memory efficient way
10  filename = "entry_1.txt"
11  data = []
12  with open(filename) as f:
13      for line in f:
14          data.append(line)
15  # remove whitespaces at end of each line
    data = [x.strip() for x in data]
```

Contents x

How to write file line-by-line?

```
l = ["pikachu", "charmander", "pidgeotto"]
fout = open("entry2.txt", "w")
for x in l:
    fout.write(x)
fout.close()
```

How to load json file? code

```
import json

file_input = "data.json"
with open(file_input) as data_file:
    data = json.load(datafile)
```

How to check if list is empty? code

```
# method 1
if not myList:
    print "list is empty"

# method 2
if len(myList) == 0:
    print "list is empty"
```

How to access index in for loop? code

```

myList = ["a", "b", "c"]

# method 1
for idx, l in enumerate(myList):
    print str(idx) + "-" + l

# method 2
idx = 0
for l in myList:
```

```
    print str(idx) + "-" + l
    idx += 1

# both methods print
# 0-a
# 1-b
# 2-c
```

Contents x

How to sort a dictionary by key alphabetically? code

```
a = {}
a["India"] = "Dhoni"
a["SouthAfrica"] = "ABD"
a["Australia"] = "Smith"

for key in sorted(a.keys(), key=lambda x:x.lower()):
    print ("{0} - {1}".format(key, a[key]))

# prints
# Australia - Smith
# India - Dhoni
# SouthAfrica - ABD
```

How to call tcl procedure in python? code

```
# let's say you have a tcl file named 'test.tcl' with contents as below.
puts "Hello"
proc sum {a b} {
    set c [expr $a + $b]
    puts "Addition of $a and $b is $c"
}
```

```
# to call a tcl proc in python, we need 'tkinter' library which comes with python us
import tkinter
r = tkinter.Tk()
r.tk.eval("source test.tcl")
r.tk.eval("sum 10 20")

# prints
# Hello
# Addition of 10 and 20 is 30
```