1, Python

Definition: Python is a high-level programming language designed to be easy to read and simple to implement. It is [open source](https://techterms.com/definition/opensource), which means it is free to use, even for commercial applications. Python can run on Mac, Windows, and Unix systems and has also been ported to [Java](https://techterms.com/definition/java) and .NET virtual machines.

Keywords: from, as, import, and, or, not, True, False, None, for, in, is, while, continue, break, pass, def, yield, return, global, nonlocal, if, else, elif, class, try, except, with, finally, raise, accept, assert, del, lambda

2,

3, Built-in function

ord()

chr()

len()

input()

map()

print()

format()

complex([real][imag])

abs()

float()

round()

int()

min()

max()

pow()

sorted()

range()

4, ***random*** module

random(): return a random real number in [0,1)

randint(a, b): return a random int in the rang [a, b]

choice(s): Choose random element from the list s

5, 6, Input and Output, format output

* Input:

input(): get the string from the keyboard

int(input()): get the int number from the keyboard

float(input()): get the float number from the keyboard

map(int, input().split()): get a line of numbers

* Output:

print(a): print object a to the screen

print

* Format output:

print(‘{:}’.format(a))

7, Conditional Operator

if x>y:

………………………

else:

………………………

Full conditional operator: if … elif … else

Ternary conditional operator: a if <condition> else b

8, Conditional Operator with multiple choices, nested conditional operator

9, Loop with condition

while <condition>:

…

[else:

…]

break: stop the loop

continue: go to the next iteration, the remaining part will be skipped

10, Loop with iterator

for <variable> in <iterable>:

…

11, math

Basic function:

acos()

asin()

cos()

sin()

degrees(): radian to degrees

ceil()

floor()

factorial()

exp()

log()

pi

e

12, String, basic functions and methods

Basic functions:

s1 + s2: concatenation of 2 strings

s\*n: n-times repeat string s

min(s): return the symbol with minimum ASCII-code

max(s): return the symbol with maximum ASCII-code

Methods:

s.center(width[, fillchar]): center a string in a field of given width

s.ljust(width[, fillchar]): left-justify a string in a file of give width

s.rjust(width[, fillchar]): right-justify a string in a file of give width

s.count(sub[, start[, end]]): count the number of occurrences of substring sub in string s

s.find(sub[, start[, end]]): return the lowest index in s where the substring sub is found

s.rfind(sub[, start[, end]]): return the highest index in s where the substring sub is found

s.strip([char]): return a copy of the string with leading and trailing characters removed

s.lstrip([char]): return a copy of the string with leading characters removed

s.rstrip([char]): return a copy of the string with trailing characters removed

s.replace(old, new[, maxreplace]): return a copy of string s with all occurrences of substring old replaced by new

13, List, creating list and list generators

Definition:

A list consists of the same or different types of elements in a given order. The size of the list can be changed over the runtime of the program and elements in the list can be duplicated.

Creating list:

To create a list, put a number of expressions in square brackets:

L = [] - empty list

L = list() - empty list

L = [expression, …] - list with initial values

L = [expression for variable in sequence]

L = list(sequence)

L = list(expression for variable in sequence)

14, List methods

L.append(obj): Appends object to list

L.extend(seq): Appends the contents of seq to list

L.insert(index, obj): Inserts object obj into the list at offset index

L.remove(obj): Removes the object obj from the list

L.count(obj): returns the number of occurrences of object obj in the list

L.index(obj): return the lowest index in the list where the obj is found

L.reverse(): Reverses objects of list in place

L.sort(): Sorts objects in the list

15, List Operators:

cmp(list1, list2): compares elements of both lists

len(list): returns the length of the list

max(list): return the maximum element of the list

min(list): return the minimum element of the list

list(seq): converts a tuple into list

16, Slices

X = [1,2,3,4,5]

X[start: end: step]

x[:] : returns a copy of the list

x[0:3]

x[1:]

x[:3]

x[:-1]

x[-3:]

x[1:-2]

17, 18, 19, 20, 21, Sort, Insertion sort, selection sort, shell sort, bubble sort, cocktail sort, quicksort

import random  
  
# Selection sort  
a = [76, 35, 12, 5, 3, 2, 1, 31, 7612]  
n = len(a)  
for i in range(n - 1):  
 for j in range(i + 1, n):  
 if a[i] > a[j]:  
 a[i], a[j] = a[j], a[i]  
print(a)  
  
# Insertion sort  
a = [76, 35, 12, 5, 3, 2, 1, 31, 7612]  
n = len(a)  
for i in range(1, n):  
 j = i - 1  
 temp = a[i]  
 while temp < a[j] and j >= 0:  
 a[j + 1] = a[j]  
 j -= 1  
 a[j + 1] = temp  
  
print(a)  
  
# Shell sort  
a = [76, 35, 12, 5, 3, 2, 1, 31, 7612]  
n = len(a)  
gap = int(1)  
while gap < n / 3:  
 gap = gap \* 3 + 1  
  
while gap > 0:  
 for i in range(gap, n):  
 temp = a[i]  
 j = i - gap  
 while temp < a[j] and j >= gap - 1:  
 a[j + gap] = a[j]  
 j -= gap  
 a[j + gap] = temp  
 gap = int((gap - 1) / 3)  
print(a)  
  
# Bubble sort  
a = [76, 35, 12, 5, 3, 2, 1, 31, 7612]  
n = len(a)  
for i in range(n - 2):  
 for j in range(0, n - i - 1):  
 if a[j] > a[j + 1]:  
 a[j], a[j + 1] = a[j + 1], a[j]  
  
print(a)  
  
# Cocktail sort  
a = [76, 35, 12, 5, 3, 2, 1, 31, 7612]  
n = len(a)  
start = 0  
end = n - 1  
swapped = True  
while swapped:  
 swapped = False  
 for i in range(start, end):  
 if a[i] > a[i + 1]:  
 a[i], a[i + 1] = a[i + 1], a[i]  
 swapped = True  
 if not swapped:  
 break  
 start += 1  
 for i in range(end, start - 1, -1):  
 if a[i] < a[i - 1]:  
 a[i], a[i - 1] = a[i - 1], a[i]  
 swapped = True  
 end -= 1  
  
print(a)  
  
# Quick sort  
a = [76, 35, 12, 5, 3, 2, 1, 31, 7612]  
n = len(a)  
  
  
def quick\_sort(a, left, right):  
 i = left  
 j = right  
 pivot = a[left + random.randint(0, right - left)]  
 while i <= j:  
 while a[i] < pivot:  
 i += 1  
 while a[j] > pivot:  
 j -= 1  
 if i <= j:  
 a[i], a[j] = a[j], a[i]  
 i += 1  
 j -= 1  
 if left < j:  
 quick\_sort(a, left, j)  
 if i < right:  
 quick\_sort(a, i, right)  
  
  
quick\_sort(a, 0, n - 1)  
print(a)

22, Set

Definition: A set object is an unordered collection of distinct hash-able objects. Every element is unique (no duplicates) and must be immutable (which cannot be changed).

Create set:

my\_set = set() – empty set

my\_set = {1, 2, 3}

my\_set = set([1, 2, 3])

Operations:

len(s): returns the number of elements in set s

s.add(x): adds object x to set s

s.discard(x): removes x from set s if present

s.remove(x): removes x from set s, raises KeyError if not present

s.clear(): removes all the elements from set s

s.pop(): removes and returns an arbitrary element from s; raises KeyError if empty

s.difference(t) (s-t): new set with elements in s but not in t

s.difference\_update(t) (s-=t): returns set s after removing elements found in t

s.intersection(t) (s&t): new set with elements common to s and t

s.intersection\_update(t) (s&=t): returns set s keeping only elements also found in t

s.issubset(t) (s<=t): check whether s is a subset of t

23, Dictionary

Definition:

Dictionary is an unordered set of key: value pairs

Keys are unique (within one dictionary) and immutable

Creating dictionary:

my\_dict = {} - empty dictionary

my\_dict = dict() - empty dictionary

my\_dict = {‘a’:’1’, ‘b’:’2’, ‘c’:’3’}

my\_dict = dict(a=1, b=2, c=3)

Methods:

dict.update(dict2): Adds dictionary dict2 to dict

dict.pop(key): Removes the key and returns the value

dict.clear(): Clears entire dictionary

d.items(): Returns a list of tuples(key, value)

24, Matrix

Definition:

A matrix in Python is defined by a list of lists

Creating matrix:

mat = [[0]\*column]\*row

mat = [[0]\*column for i in range(row)]

Input:

Input 1 element at a time

Input 1 line at a time:

mat = [[int(x) for x in input().split()] for i in range(row)]

Output:

print(mat)

Operations: swap 2 columns, swap 2 rows, main-diagonal

25,

26, Function

Definition:

A function is a block of organized, reusable code that is used to perform a single, related action.

Define a function:

def function\_name(parameters):

function\_suite

return [expression]

Function Arguments:

* Required arguments

def printme(str):

print str

return;

printme(str)

* Keyword arguments

def printme(str):

print str

return;

printme(str = "My string")

* Default arguments

def printme(str = “My string”):

print str

return;

printme()

* Variable-length arguments

def printinfo(arg1, \*vartuple):

for var in vartuple:

print var

return

The return Statement

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None

27, Lambda and Recursive function

The Anonymous Functions

These functions are called anonymous because they are not declared in the standard manner by using the def keyword.

Syntax

lambda [arg1[, arg2, … argN]]: expression

Example

sum = lambda arg1, arg2: arg1 + arg2

print(sum(10, 20))

Recursive function

In python, a function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions

28, Text file, open file, mode of access to the file

Text file:

Text files are structured as a sequence of lines, where each line includes a sequence of characters. Each line is terminated with a special character, call the End of Line character.

Open() function:

**file\_object = open(“filename”, “mode”)**

Mode:

r – read mode which is used when the file is only being read

w – write mode which is used to edit and write new information to the file

a – appending mode which is used to add new data to the end of the file

r+ - special read and write mode, which is used to handle both actions when working with a file

29, read, write, search in file

Read:

file = open(‘…’,’r’)

file.read() # extract a string that contains all characters in the file

file.readline() # extract a string that contains a single line from the file

file.readlines() # returns a list of every lines in the file

Write:

file.write(‘…’) # append … to the end of the file

Looping over a file object:

For line in file:

30,

31, Binary files, data serialization, module pickle

file\_read = open(‘db.bin’, ‘rb’)

my\_list = pickle.load(file\_read)

# read a pickled object representation from the open file object file\_read

file\_read.close()

file\_write = open(‘db.bin’, ‘wb’)

pickle.dump(my\_list, file\_write)

# write a pickled representation of my\_list to the open file object file\_write

file\_write.close()

32, numpy, array

Definition:

Numpy is a Python package. It stands for ‘Numerical Python’. It is a library consisting of multidimensional array objects and a collection of routines for processing of array

Array:

* Create array:

import numpy as np

a = np.array([1, 2, 3])

a = np.full((3, 4), 4)

* Indexing:

Slicing: Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array

import numpy **as** np

*# [[ 1 2 3 4]*

*# [ 5 6 7 8]*

*# [ 9 10 11 12]]*

a **=** np**.**array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

*# [[2 3]*

*# [6 7]]*

b **=** a[:2, 1:3]

*# A slice of an array is a view into the same data, so modifying it*

*# will modify the original array.*

**print**(a[0, 1]) *# Prints "2"*

b[0, 0] **=** 77 *# b[0, 0] is the same piece of data as a[0, 1]*

**print**(a[0, 1]) *# Prints "77"*

Integet array indexing:

import numpy **as** np

a **=** np**.**array([[1,2], [3, 4], [5, 6]])

**print**(a[[0, 1, 2], [0, 1, 0]]) *# Prints "[1 4 5]"*

*# The above example of integer array indexing is equivalent to this:*

**print**(np**.**array([a[0, 0], a[1, 1], a[2, 0]])) *# Prints "[1 4 5]"*

*# When using integer array indexing, you can reuse the same*

*# element from the source array:*

**print**(a[[0, 0], [1, 1]]) *# Prints "[2 2]"*

*# Equivalent to the previous integer array indexing example*

**print**(np**.**array([a[0, 1], a[0, 1]])) *# Prints "[2 2]"*

a[[list row],[list column]]

33, numpy

np.add

np.subtract

np.multiply

np.divide

np.max

np.min

34, matplotlib, building graphics and controlling drawing area

**import** **matplotlib.pyplot** **as** **plt**

plt.plot([1,2,3,4], [1,4,9,16], 'ro')

# plotting to canvas

plt.axis([0, 6, 0, 20])

plt.show()

# showing what we plotted

35, matplotlib, building histogram and piechart

Histogram:

population\_ages = [22, 55, 62, 45, 21, 22, 34, 42,43, 4, 99, 102, 121, 122, 130, 111, 115, 112, 80, 75, 65, 54, 44]

bins = [0,10,20,30,40,50,60,70,80,90,100,110,120,130]

plt.hist(population\_ages, bins, histtype=’bar’)

Piechart:

slices = [7,2,2,13]

activities = [‘sleeping’, ‘eating’, ‘working’, ‘playing’]

cols = [‘c’, ’m’, ’r’, ’k’]

plt.pie(slices, labels=activities, colors=cols, startangle=90)