Python

Python uses indentation instead of semicolons

Could be run directly

On terminal, enter “python” enters python 2, “python3” enters python 3.

Python3 <py file> to run

Python3 –version to check version

First line of comment: shebang line for unix

Start with #! Path to executable and intepretor

#!/usr/local/env python3

Define Function:

Could be just defined by x=, no need for int/float/str

Def function():

#anything in here would be defined locally within function()

X=0

#globalize

global x

Function:

<function name>(arg1, arg2,…)

function(\*<argument>), \* specifies that there could be multiple entries of arguments

\*\*kwarg for key arguments, like <name>=<arg>

Use kwarg for keyarguments

def function(\*<argument>):

for x in <argument>:

x = x+r

Print:

print(<content>) to print

use “” to mark string,

to combine, use print(<content1> + <content2>)

the types can’t be different, so to convert function to string, use str(<function>) to typecast

Or print(<content1>,<content2>)

Print(“balabhab {}” .format(x))

X will follow print

{} .format(x) will make x into the {}

Legacy mode is %<placeholder> then % in the end

Typecast:

Type(<function>) returns type of function, in class int, float or str

To typecast, use

Int(<function>)

Float(<function>)

Str(<function>)

String: use ‘’’ to make multi strings

.capitalize() capitalizes

.upper makes upper case

.lower makes lower case

{1:<09}[0:<09] 1:< makes 9 space to the right, 0 makes 9 spaces to the left, 9 is how many spaces in total and 0 preceding 9 means fill empty space with 0

Number Operation:

// divides without remaining.

Sacrifices accuracy for precision

For money, from decimal import \*

A=Decimal(“.10”)

Boolean:

Has True and False

Comparison operator returns True or False in Bool

None is NoneType

None, empty string and 0 is false, everything else is true

List and Tuple:

[] marks list

() marks Tuple

Tuple is not mutable later

Range(5) is 0-4

Range(x,y) is range from x to y-1

Range(x,y,z) is range from x to y-1 in z increment

Tuple =(“a”,”b”,…) or

List= [(‘lastn1’, ‘firstn1’, ‘number1’), (‘lastn2’, ‘firstn2’, ‘number2’)]

List[0] would be “a” or (‘lastn1’,’firstn1’,’number1’)

List[1:2:3] returns index 1, 2, 3 item of the list

<listname>.index(“name”) returns the index number of the “name”

<listname>.append(“name”) appends the “name” to the end of the list

<listname>.insert(<index>,”name”) inserts the “name” before the indexed item

<listname>.remove(“name”) removes the “name”

<listname>.pop() removes the last item and returns the “name”

<listname>.pop(<index>) removes the indexed item

Lens(<list>) gives the length of the list

Print\_list() prints the list

List in a list <listname>[<first index>][<second index>]

List Comprehension:

DIctonary:

Defined by {<key>:<value>, <key>:<value>}

Or by dict(<key>=<value>, <key>=<value>)

String or number can be keys

<name>.items() returns the keys and values in pairs

<name>.keys() returns the keys

<name>.values() returns the values

<name>[key] returns the value

<name>.get(<key>) returns value if key exists, and prevents error when key doesn’t. Returns None instead.

Set:

Set()

Set is just a set of the symbols and characters in the set

Id returns unique id, type returns specific type

Int has same id, same strings are different ids

Isinstance(<function>, <type>) returns true or falses

Arithmetic operator

+ Addition

- subtraction

\*multiplication

/ divison

// interger divison

% remainder

\*\* exponent

-unary negative

+unary positive

Bitwise operators

& and

| or

^ Xor

<< shift left

>> shift right

Boolean operators

And

Or

Not

In Value in set

Not in Value not in set

Is

Is not

Input:

<function name> = input (<prompt>)

Default input in string, needs typecast

Conditional:

If (<condition1>):

Blahblahblah

Elif(<condition2>): 🡨 something that is not condition 1 but fulfills condition 2

Blahblahblah

Else: 🡨neither condition 1 nor condition 2 is true

blahblah

also can be

<variable> = blabhl if (<condition1>) else blabhabl

Loop:

Only while and for

def main():  
 x = 0  
  
 # define a while loop  
 while (x<5):  
 print(x)  
 x = x+1  
  
  
 # define a for loop  
 for x in range(5,10):  
 print(x)  
  
 # use a for loop over a collection  
  
 days=["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]  
 for d in days:  
 print(d)

break breaks the loop

continue jumps back to top of loop

For runs in the list till end

enumerate(iterable, start=0)

i is index from enumerate, d is object

days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]  
for i,d in enumerate(days,1):  
 print(i,d)

enumerate(d, 1) returns

(1, 'Mon')

(2, 'Tue')

(3, 'Wed')

(4, 'Thu')

(5, 'Fri')

(6, 'Sat')

(7, 'Sun')

This is what the enumerate turns the data into

Classes:

Class <classname>(<inherited class>)

class myClass():  
 def method1(self):  
 print("myClass method1")  
  
 def method2(self, someString):  
 print("myClass method2 " + someString)  
  
class anotherClass(myClass):  
 def method1(self):  
 myClass.method1(self)  
 print("anotherClass method1")  
  
 def method2(self, someString):  
 print("anotherClass method2 " + someString)  
  
  
def main():  
 c = myClass()  
 c.method1()  
 c.method2("This is a string")  
  
 c2 = anotherClass()  
 c2.method1()  
 c2.method2("This is a string")

How to construct class

class Animal:  
 def \_\_init\_\_(self, type, name, sound): #initializer  
 self.\_type = type  
 self.\_name = name  
 self.\_sound = sound  
  
 def type(self): #differentiate variables from getters  
 return self.\_type  
  
 def name(self):  
 return self.\_name  
  
 def sound(self):  
 return self.\_sound

The \_ makes the variable private and not importable from the class.

Or using kwargs, so argument sequence doesn’t matter.

class Animal:  
 def \_\_init\_\_(self, \*\*kwargs): #initializer  
 self.\_type = kwargs["type"]  
 self.\_name = kwargs["name"]  
 self.\_sound = kwargs["sound"]  
  
 def type(self): #differentiate variables from getters  
 return self.\_type  
  
 def name(self):  
 return self.\_name  
  
 def sound(self):  
 return self.\_sound

def \_\_str\_\_(self):  
 return f'The {self.type()} is named "{self.name()}" and says "{self.sound()}"

the \_\_str\_\_\_ enables the class to be printable by itself

When defined by def, the variable is of the object, so by changing the object’s variable value doesn’t change the variable of other objects of the same class.

Dates and time:

from datetime import date  
from datetime import time  
from datetime import datetime

# Get today's date from the simple today() method from the date class  
  
today = date.today()  
print(today)  
  
# print out the date's individual components  
  
print(today.day, today.month, today.year)  
  
# retrieve today's weekday (0=Monday, 6=Sunday)  
  
print(today.weekday())  
days = ["a", "b","c", "d", "e", "f", "g"]  
print(days[today.weekday()])

## DATETIME OBJECTS  
# Get today's date and time from the datetime class  
  
today = datetime.now()  
print (today)  
  
# Get the current time  
t = datetime.time(datetime.now())  
print (t)

Using String format time:

Time value can be embedded in the string

from datetime import datetime  
  
def main():  
 # Times and dates can be formatted using a set of predefined string  
 # control codes  
  
 now = datetime.now()  
  
  
 #### Date Formatting ####  
   
 # %y/%Y - Year, %a/%A - weekday, %b/%B - month, %d - day of month  
  
 print(now.strftime("The current year is %a, %d, %B, %y"))  
  
  
 # %c - locale's date and time, %x - locale's date, %X - locale's time  
 print (now.strftime("Locale date and time %c"))  
 print (now.strftime("Locale date %x"))  
 print (now.strftime("Locale time %X"))  
  
 #### Time Formatting ####  
   
 # %I/%H - 12/24 Hour, %M - minute, %S - second, %p - locale's AM/PM  
  
 print(now.strftime("Current time: %I:%M:%S %p"))  
 print(now.strftime("24hr time: %H:%M:%S "))

Result:

The current year is Tue, 31, March, 20

Locale date and time Tue Mar 31 17:36:14 2020

Locale date 03/31/20

Locale time 17:36:14

Current time: 05:36:14 PM

24hr time: 17:36:14

Time Delta:

from datetime import date  
from datetime import time  
from datetime import datetime  
from datetime import timedelta  
from datetime import tzinfo  
  
def main():  
  
 # construct a basic timedelta and print it  
  
 print(timedelta(days=365, hours=5, minutes=1))  
  
 # print today's date  
  
 now = datetime.now()  
 print("today is:" + str(now))  
  
 # print today's date one year from now  
  
 print("one year from now it will be:" + str(now+timedelta(days=365)) )  
  
  
 # create a timedelta that uses more than one argument  
  
 print("In 2 days and 3 weeks, it will be:" + str(now+timedelta(days=2, weeks=3)))  
  
 # calculate the date 1 week ago, formatted as a string  
 t = datetime.now() - timedelta(weeks=1)  
 s = t.strftime("%A %B %d %Y")  
 print( "One week ago it was: " + s)  
  
### How many days until April Fools' Day?  
  
 t= date.today()  
 afd=date(t.year, 4, 1)  
  
# use date comparison to see if April Fool's has already gone for this year  
# if it has, use the replace() function to get the date for next year  
  
 if afd<t:  
 print ("April Fool's day already went by by %d days ago" % (t-afd).days)  
 afd = afd.replace(year= today.year+1)  
  
  
# Now calculate the amount of time until April Fool's Day   
  
 time\_to\_afd = afd-t  
 print("It's just %d day until April Fool's day" % time\_to\_afd.days )  
  
# days from a given day  
  
 givenday = datetime(2020, 3, 6, 20, 30, 50)  
 print("Time 1 week 2 days 1 minute and 20 seconds from 2020-03-06 20:30:50 is "  
 + str(givenday+timedelta(days=2,weeks=1,minutes=1,seconds=20)))

Result

365 days, 5:01:00

today is:2020-03-31 18:29:22.874879

one year from now it will be:2021-03-31 18:29:22.874879

In 2 days and 3 weeks, it will be:2020-04-23 18:29:22.874879

One week ago it was: Tuesday March 24 2020

It's just 1 day until April Fool's day

Time 1 week 2 days 1 minute and 20 seconds from 2020-03-06 20:30:50 is 2020-03-15 20:32:10

Calendar:

# import the calendar module  
  
import calendar  
  
# create a plain text calendar  
  
c = calendar.TextCalendar(calendar.MONDAY)  
st = c.formatmonth(2017, 1, 0, 0)  
print(st)  
  
  
  
# create an HTML formatted calendar  
  
hc = calendar.HTMLCalendar(calendar.SUNDAY)  
st = hc.formatmonth(2017,1,0)  
print(st)  
  
# loop over the days of a month  
# zeroes mean that the day of the week is in an overlapping month  
  
for i in c.itermonthdays(2017,8):  
 print(i)  
   
# The Calendar module provides useful utilities for the given locale,  
# such as the names of days and months in both full and abbreviated forms  
  
for name in calendar.month\_name:  
 print(name)  
  
print("\n")  
  
for day in calendar.day\_name:  
 print(day)  
  
# Calculate days based on a rule: For example, consider  
# a team meeting on the first Friday of every month.  
# To figure out what days that would be for each month,  
# we can use this script:  
  
print ("Team meetings will be on: ")  
for m in range(1,13):  
 cal = calendar.monthcalendar(2020,m)  
 wkone=cal[0]  
 wktwo=cal[1]  
  
 if wkone[calendar.FRIDAY]!=0:  
 meetday = wkone[calendar.FRIDAY]  
 else:  
 meetday = wktwo[calendar.FRIDAY]  
  
 print("%10s %2d" % (calendar.month\_name[m], meetday))

Read and write txt file:

def main():   
 # Open a file for writing and create it if it doesn't exist  
 f=open("textfile.txt", "w+")  
  
  
 # Open the file for appending text to the end  
 f = open("textfile.txt", "a")  
   
 # Open the file to read  
 f=open("textfile.txt", "r")  
  
 # write some lines of data to the file  
 for i in range(10):  
 f.write("This is line" + str(i) + "\r\n")  
   
 # close the file when done  
 f.close()  
  
   
 # Open the file back up and read the contents  
 if f.mode == 'r':  
 # contents = f.read()  
 # print(contents)  
 fl=f.readlines()  
 for x in fl:  
 print(x)

note there are two different ways of reading, one by line and the other as a whole.

Show file directory and last modified time:

import os  
from os import path  
import datetime  
from datetime import date, time, timedelta  
import time  
  
  
def main():  
 # Print the name of the OS  
 print(os.name)  
  
 # Check for item existence and type  
 print("Item exists: " + str(path.exists("textfile.txt")))  
 print("Item is a file: " + str(path.isfile("textfile.txt")))  
 print("Item is a directory: " + str(path.isdir("textfile.txt")))  
  
 # Work with file paths  
 print("Item path: " + str(path.realpath("textfile.txt")))  
 print("Item path and name: " + str(path.split(path.realpath("textfile.txt"))))  
  
 # Get the modification time  
 t=time.ctime(path.getmtime("textfile.txt"))  
 print(t)  
 print(path.getmtime("textfile.txt"))  
 print(datetime.datetime.fromtimestamp(path.getmtime("textfile.txt")))  
   
 # Calculate how long ago the item was modified  
 td=datetime.datetime.now() - datetime.datetime.fromtimestamp(  
 path.getmtime("textfile.txt")  
 )  
 print ("It has been " + str(td) + " since the file has been modified")  
 print ("Or, " + str(td.total\_seconds()) + " seconds")

datetime.datetime.utcfromtimestamp utc timestampo from posix time in seconds

path.realpath gives the path and name of file

path.split splits given path into path and file

posix

Item exists: True

Item is a file: True

Item is a directory: False

Item path: /Users/zhaohan\_dong/Desktop/python/Ex\_Files\_Learning\_Python/Exercise Files/Ch4/textfile.txt

Item path and name: ('/Users/zhaohan\_dong/Desktop/python/Ex\_Files\_Learning\_Python/Exercise Files/Ch4', 'textfile.txt')

Thu Apr 2 14:03:15 2020

1585850595.8619256

2020-04-02 14:03:15.861926

It has been 0:38:10.860946 since the file has been modified

Or, 2290.860946 seconds

Copy file and zip:

import os  
from os import path  
import shutil  
from shutil import make\_archive  
from zipfile import ZipFile  
  
  
def main():  
 # make a duplicate of an existing file  
 if path.exists("textfile.txt"):  
 # get the path to the file in the current directory  
 src=path.realpath("textfile.txt")  
   
 # let's make a backup copy by appending "bak" to the name  
 dst=src+".bak"  
   
 # copy over the permissions, modification times, and other info  
 shutil.copy(src,dst)  
 # copy over the metadata (mod time...)  
 shutil.copystat(src,dst)  
  
 # rename the original file  
 os.rename("textfile.txt", "newfile.txt")  
   
 # now put things into a ZIP archive  
 root\_dir, tail=path.split(src)  
 shutil.make\_archive("archive", "zip", root\_dir)  
  
 # more fine-grained control over ZIP files  
 with ZipFile("testzip.zip","w") as newzip:  
 newzip.write("textfile.txt")  
 newzip.write("textfile.txt.bak")

schutil.copy(<original file>,<newfile with extension>) copies permission

schutil.copystat copies metadata

Return vs Yield

Yield is a generator which doesn’t store the information

Return returns the value from the function

Decorator:

@<function name> followed directly by

<wrapped function name>

So the wrapped function becomes the argument in the @ function

Useful when calculating elapsed time, so the function can be wrapped in an elapsed time function and called by the function name, not the elapsed time function name