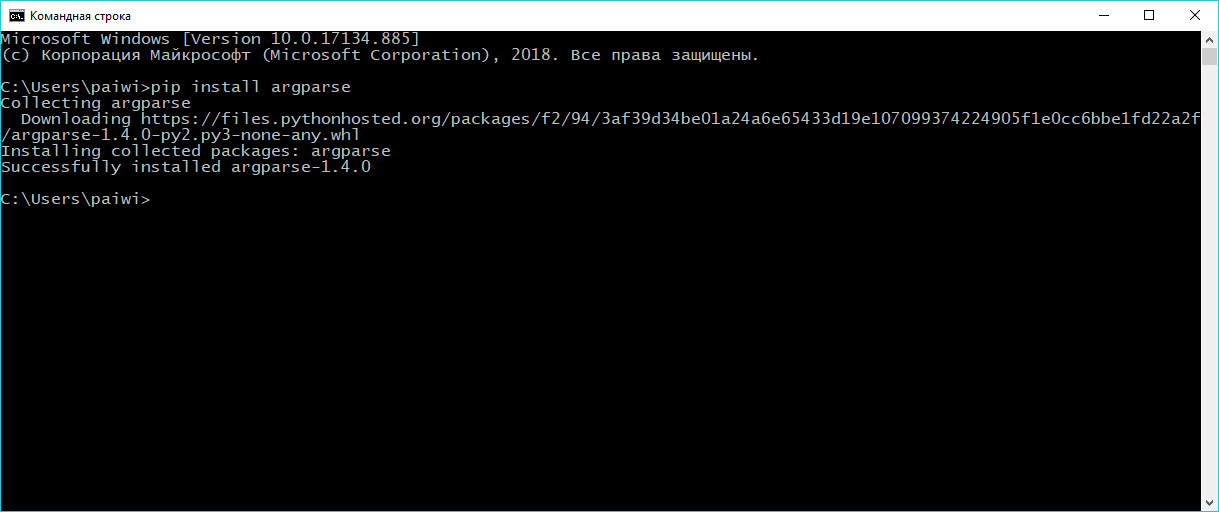
1. Напишите программу, демонстрирующую содержимое каталогов. Программа должна иметь следующие интерфейс:

ls.py [option] path

Options:

* 1. -H, --hidden – показывает скрытые файлы (по умолчанию false). Скрытые файлы – это файлы начинающиеся с ‘.’.
  2. -m, --modified – показывает дату создания файла/каталога;
  3. -o ORDER, --order=ORDER – сортирует вывод по заданному полю.  
     ORDER:
     1. n – сортировка по имени (она же по умолчанию);
     2. s – сортировка по размеру.
  4. -s, --size – вывод размера всех файлов.

Для считывания аргументов из командной строки установим сторонний пакет “argparse”. Перейдем в командную строку и воспользуемся менеджером пакетов pip.



Далее перейдем в IDE и начнем разработку приложения.

# Импортируем пакет argparse  
import argparse  
# Импортируем стандартный модуль os  
import os  
# Импортируем подмодуль datetime из модуля datetime  
from datetime import datetime  
  
# Пакет argparse предназначен для работы с командной строкой  
# Создаем переменную parse с помощью функции ArgumentParser() из пакета argparse  
parse = argparse.ArgumentParser()  
# Добавляем аргументы, которые программа ожидает от пользователя с помощью функции add\_argument переменной parse  
# --path - переменная которая хранит путь до директории, содержимое которой программа должна вывести на экран консоли  
# type - этот параметр указывает к какому типу требуется привести полученное значение из командной строки  
# default(значение по умолчанию) - это значение которое подставиться, если пользователь не укажет этот параметр  
# при запуске программы.  
# По умолчанию программа выводит содержимое папки в котором запускается файл ls.py  
parse.add\_argument('--path', type=str, default='./')  
# -H - это короткое имя ключа т.е псевдоним для ключа --hidden  
# По умолчанию программа не выводит на консоль файл, имя которых начинается с "."  
parse.add\_argument('-H', '--hidden', type=bool, default=False)  
# -m, --modified - этот ключ отвечает за вывод даты создания файла  
parse.add\_argument('-m', '--modified', type=bool, default=False)  
# -o, --order - ключ, который отвечает за порядок вывода списка файлов и директорий внутри заданной директории  
# Парметр choice - описывает список элементов, который возможен в качестве получаемого значения  
# По умолчанию список выводится отсортированным по имени файлов  
parse.add\_argument('-o', '--order', type=str, choices=['s', 'n'], default='n')  
  
# После описания всех допустимых параметров, требуется вызвать функцию parse\_args() перемнной parser  
args = parse.parse\_args()  
# На основе полученных значений составим словарь set\_args, который будет содержать значение всех допустимых параметров командной строки  
# Для получения этих значений обратимся к переменной args по следующему шаблону: args.имя\_параметра  
set\_args = {  
 # Ключ path хранит путь до директории список элементов которой требуется вывести на консоль  
 'path': args.path,  
 # Ключ hidden хранит булевскую переменную, отвечающую за вывод скрытых файлов  
 'hidden': args.hidden,  
 # Ключ modified тоже булевского типа и отвечает за вывод даты создания файла  
 'modified': args.modified,  
 # Ключ order хранит строку, которая указывает какой тип сортировки требуется применить  
 'order': args.order  
}  
  
  
# Этот метод предназначен для сортировки списка элементов рассматриваемой директории  
# val - это элемент списка, который сам является списком и содержит максимум 3 элемента:  
# val[0] - имя файла/директории  
# val[1] - это дата создания файла если -m при запуске программы указан как True, или  
# это размер файла в байтах, если ключ -o использован вместе со значением s  
# val[2] - это размер файла в байтах, если ключ -o использован вместе со значением s  
# и ключ -m указан как True  
def val\_by\_sort(val):  
 # Проверка на то, было ли значение ключа -o указано как s. Если True, то возвращаем размер файла  
 # -1 - это обращение к последнему элементу в списке  
 if set\_args['order'] == 's':  
 return val[-1]  
 # Иначе возвращаем имя файла  
 else:  
 return val[0]  
  
  
# Проверяем существует ли указанный путь и что указанный путь - это директория, а не файл  
if os.path.exists(set\_args['path']) and os.path.isdir(set\_args['path']):  
 # Объявляем список, в который будем сохранять имя файла и дополнительную информацию о нём  
 dir\_content = []  
 # функция os.listdir() - возвращет список имен элементов внутри заданной директории  
 for item in os.listdir(set\_args['path']):  
 # Проверяем, требуется ли добавлять файлы, имена которых начинаются с точки  
 # Для проверки имени файла воспользуемся функцией startswith(), которая проверяет начинается ли строка  
 # с заданной строки  
 if not set\_args['hidden'] is True and item.startswith('.'):  
 # Если условие выполнилось, тогда пропускаем эту итерацию и переходим к следующей итерации  
 continue  
 else:  
 # Создаем список с именем файла  
 lst\_items = [item]  
 # Получаем полный путь до текущего файла  
 path\_to\_item = os.path.join(set\_args['path'], item)  
 if set\_args['modified']:  
 # Получаем время в секундах, когда был изменен файл  
 m\_time = os.path.getctime(path\_to\_item)  
 # Переводим секунды в формат: год-месяц-день час-минута-секунда  
 format\_time = datetime.fromtimestamp(m\_time).strftime('%Y-%m-%d %H:%M:%S')  
 lst\_items.append(format\_time)  
 if set\_args['order'] == 's':  
 # Получаем размер файла в килобайтах  
 kb\_size = os.path.getsize(path\_to\_item)  
 lst\_items.append(os.path.getsize(path\_to\_item))  
 dir\_content.append(lst\_items)  
 # Сортируем получившийся список по заданному условию, для этого воспользуемся функцией sort  
 # Функция sort принимает параметр key, который отвечает за поле по которому будет отсортирован список   
 dir\_content.sort(key=val\_by\_sort)  
 # Красиво выводим на консоль результат  
 for f\_list in dir\_content:  
 file = f\_list[0]  
 print(file, end='\t')  
 for item in f\_list[1:]:  
 print(item, end='\t')  
 print()  
else:  
 # Если передали не правильный путь, тогда сообщаем об этом пользователю  
 print('Ожидалась существующая директория')

Пример запуска файла ls.py:

ls.py -m True -o s

\_\_pycache\_\_ 2019-06-24 22:56:40 0

abc.txt 2019-07-04 21:56:47 15

test.txt 2019-06-30 16:14:44 55

data.bin 2019-07-07 19:12:35 117

students.csv 2019-07-04 22:32:51 181

balances.xlsx 2019-07-07 17:55:48 4824

ls.py 2019-06-24 20:57:33 8463

1. Напишите программу, которая получает на вход от пользователя скорость (м/с) и угол броска (в градусах). Программа по входным данным должна рассчитать дистанцию (в метрах), которую преодолеет брошенное тело.

Решение:

# Импортируем модуль math  
import math  
  
# Ускорение свободного падения  
g = 9.8  
# Задаём скорость с которой брошено тело  
v0 = float(input("Введите начальную скорость: "))  
# Задаём угол, под которы брошено тело  
alpha = float(input("Введите угол броска: "))  
# Переводим градусы в радианы  
alpha\_grad = alpha \* math.pi / 180  
# Заранее считаем sin(2\*alpha)  
sin2alpha = math.sin(alpha\_grad \* 2)  
# Возводим в квадрат начальную скорость  
v0\_square = v0 \*\* 2  
# Рассчитываем максимальную длину броска  
L = v0\_square \* sin2alpha / g  
print("Результат: ", L)

Результат работы программы:

Введите начальную скорость: 10

Введите угол броска: 40

Результат: 10.049058704206203

Задания для самостоятельного решения:

1. Дополните программу 1:
   1. Программа должна выводить количество файлом и директорий внутри проверяемой директории.
   2. Размер файлов и директорий должен выводиться в мегабайтах.
   3. Найти все файлы и директории, начинающиеся с заданной подстроки. Параметр должен быть получен из командной строки
2. Рассчитать общее время движения тела, брошенного под углом к горизонту.
3. Рассчитать максимальную высоту подъема тела, брошенного под углом к горизонту,