Курс по Python

Урок №1

“Присваивание”

# рассмотрим присваивание значений в Python  
# присваивание происходит последовательно справа налево  
x = y = 0  
print(x)  
print(y)  
  
y1 = 5  
x1 = y1 = 7  
  
print(x1)  
print(y1)

Урок №2

“Логический тип”

# рассмотрим логический тип данных  
# возвращает true или false  
print(13 == 13)  
  
# в Python присутствует сножественное сравнение  
x = 2  
print(1 < x < 3)  
  
# логические операторы  
# логические операторы ленивые если возможно  
# вернуть раньше значение из функции, чем закончится выполнение выражения  
# то возвращаем результат  
x = False  
y = True  
  
print(x and y or x and (not y))  
print(False and True)  
  
# задача  
year = 2017  
is\_leap = year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)  
print(is\_leap)

Урок №3

Строки

# рассмотрим работу со строками как их задавать  
# и так же как их экранировать  
  
# Сырые строки обозначаются r  
# и убирают экранирование символов, оставляя их как  
# просто символы в строке  
example\_str = r"C:\\work\\"  
print(example\_str)  
  
# строки можно переносить  
# для этого используем обратный слеш  
str2 = "this text is this text " \  
 "this text is this text" \  
 "this text is this text"  
print(str2)  
  
# для написания большого блока кода  
  
str3 = """  
 this text is this text   
 this text is this text  
 this text is this text  
"""  
print(str3)  
  
# строки можно объединять  
# строки неизменяемые при сложении двух строк  
# создаётся новый объект в памяти  
# это можно узнать по адресу в памяти  
str4 = "str4"  
print(id(str4))  
str4 += 'a'  
print(id(str4))  
  
# срезы строк  
# [start:stop:step]  
# срез может идти справа налево или  
# слева направо главное помнить что он начнётся с элемента  
# start, по умолчанию шаг 1, если указать только шаг -1  
# то массив перевернётся  
  
example\_str = "Курс по Python на Coursera"  
  
print(example\_str[9:])  
print(example\_str[9:15])  
print(example\_str[-8:-2])  
print(example\_str[::-1])

# методы строк  
quote = "покажите мне код"  
  
# метод нахождения колличества букв в строке  
print(quote.count('о'))  
  
# сделать первую букву большой  
print(quote.capitalize())  
  
# позволяет проверить является ли строка числом  
tempStr = '2017'  
print(tempStr.isdigit())  
  
# оператор in позволяет проверить содержится ли  
# подстрока в строке  
tempStr = 'Hello'  
print('lo' in tempStr)  
  
# преобразования строк  
num\_string = str(999.01)  
print(num\_string, type(num\_string))  
  
# иттерация по строке  
for elem in num\_string:  
 print(elem, end="")

# форматирование строк  
  
# передаём placeholder  
str = "{} не лгут, а лгут {}".format('Цифры', "люди")  
print(str)  
  
# передам именованный placeholder  
# неименованные аргументы указать уже нельзя  
problem = "задач"  
str = "{num} Кб должно хватить для {problem}".format(  
 num = 640, problem=problem)  
print(str)  
  
# f-строки  
subj = "optimize"  
auth = 'Knuth'  
  
print(f"Преждевременная {subj} - плохо. Автор {auth}")  
  
  
# вывод чисел с модификаторами  
num = 2 / 3  
print(f"{num:.3f}")  
print("{num:.3f}".format(num=num))

# ввод и вывод строк  
#nameUser = input()  
#print(nameUser)  
  
# байтовые строки  
# Русские символы не поддерживает  
example\_str = b"hello"  
print(example\_str, type(example\_str))  
  
for elem in example\_str:  
 print(elem, end=" ")  
print()  
  
# метод encode для строки  
# поддерживает Русские символы  
# каждый символ кодируется двумя буквами  
example\_str = "привет"  
  
example\_str = example\_str.encode()  
print(example\_str)  
  
# метод decode  
example\_str = example\_str.decode()  
print(example\_str)

Урок №4

None

# рассмотрим особый тип None  
# его булевый тип всегда False как и у пустой строки  
income = None  
  
if income is None:  
 print("Ещё даже не продавали")  
elif not income:  
 print("Ничего не заработали")  
  
print(None is income)  
print((bool(income)))

Урок №5

Управление потоком

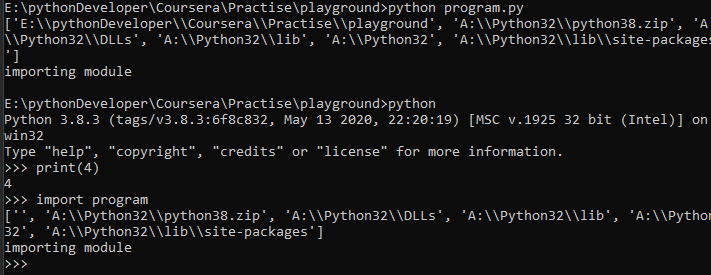
# управление потоком  
  
# после первого выполненного блока, выходит их цикла  
company = 'google.com'  
  
if 'com' in company:  
 print("good")  
elif 'goog' in company:  
 print('good1')  
else:  
 print('bad')  
  
# аналог тернарного оператора  
winner = 'Argentina' if 5 > 4 else 'Jamaica'  
print(winner)  
  
# оператор while  
indexLoop = 0  
while(indexLoop < 10):  
 indexLoop += 1  
  
# встроенный range  
# иттерируемся по целым числам  
# последнее число не входит  
# range(start, stop, step)  
for i in range(3):  
 print(i, end=" ")  
  
print()  
for i in range(1, 10, 2):  
 print(i, end=" ")  
  
print()  
for i in range(-5, -1, 1):  
 print(i, end=" ")

Урок №6

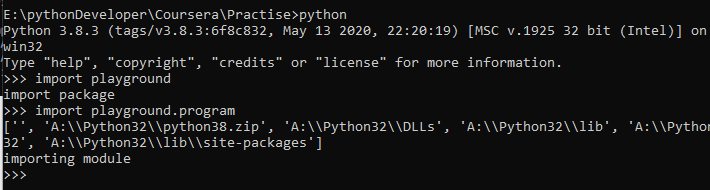
Модули и пакеты

import sys  
  
# чтобы узнать где python будет искать библиотеки   
# при импорте  
print(sys.path)

1. В python файлы можно вызывать из консоли, как в примере выше, но также можно вызывать файлы, как модули (пакеты), как показано во втором примере. При этом весь код, который может выполниться внутри файла будет выполнен сразу.



2. В python можно создать пакет, где в пакете находится множество модулей, так же в данном пакете обязательно должен быть файл \_\_init\_\_.py , который выполнится при импорте пакета, модули из пакета импортируются через точку.



3. В python код не выполняется напрямую из файла, сначала он преобразуется в оптимизированный байт код.

4. if \_\_name\_\_ == “\_\_main\_\_”:

<код>

Данный код позволяет выполнять код только, если он вызван, как главный фай, а не из модуля

5. Чтобы вызвать функцию из другого модуля используем псевдоним

import playground.utils as funck  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print('main')  
 print(funck.multiply(2, 3))  
  
print("import is pack")

6. Чтобы импортировать функцию так же можно использовать синтаксис

from playground.utils import multiply  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print('main')  
 print(multiply(2, 3))  
  
print("import is pack")

7.

import inspect  
import os  
  
# чтобы посмотреть где находятся модули в операционной системе  
# аргумент это имя модуля, который уже импортирован  
print(inspect.getfile(inspect))  
  
# чтобы посмотреть все файлы в директории  
# используем путь и следующую команду  
# где в качестве аргумента указываем путь  
print(os.listdir("A:/Python32/lib/"))

Урок №7

Виртуальное окружение

1. Для удобной работы с библиотеками python, перед началом работы (создания проекта) создаём виртуальное окружение.



2. Можно установить jupyter notebook, в него так же входит редактор кода iPython

pip install kupyter

2.1 Для запуска активируем окружение куда установили jupyter, а и прописываем команду

jupyter-notebook

Урок №8

Объектная структура и байткод

В Python всё является объектами, каждый объект имеет счётчик ссылок и тип хранимого объекта.

С помощью встроенных структур от C можно создавать свои собственные типы.

Урок №9

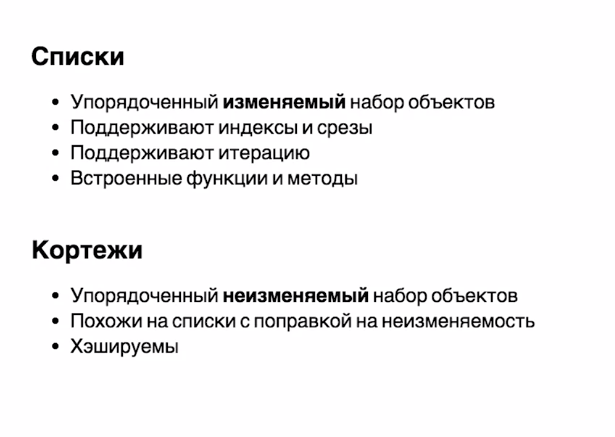
Списки и кортежи

# python коллекции  
  
# рассмотрим список и его методы  
# данными способами можно инициализировать list  
empty\_list = []  
empty\_list = list()  
user\_list = [1, ['name', True], 23.45]  
  
# срез списка (последний элемент  
# не входит в список  
# при получени среза получается новый объект  
print(user\_list[1:2])  
  
# добавление и расширение списков  
tempList = ["new", "list"]  
user\_list.extend(tempList) # расширение  
print(user\_list)  
  
tempList.append(5) # добавление  
print(tempList)  
  
tempList += 'a' # ущё один способ добавления  
  
# метод для форматирования списков join  
print(", ".join([str(elem) for elem in tempList]))  
  
# сортировка в list  
# list.sort сортирует исходный список возвращвет None  
# sorted(list) возвращает отсортированную копию списка  
tempSort = [1, 3, 2, 0, 8, 6]  
  
print(sorted(tempSort, reverse=True)) # вернулась копия  
  
tempSort.sort(reverse=True) # обратный порядок сортировки  
print(tempSort)

№9.1

Кортежи

# кортежи  
# кортежи имеют небольшой выйгрыш по скорости  
# по сравнению с list, но незначительную поэтому лучше использовать list  
  
testTuple = tuple()  
testTuple = ()  
  
# кортежи неизменяемые  
# в них нельзя добавить ничего нового  
# но если в картеже есть массив, то в массив внутри  
# кортежа можно добавить элементы  
immutable = (int, str, tuple)  
  
# у кортежей есть метод hash  
# если внктри кортежа есть изменяемы структуры  
# то мы не сможем взять метод hash  
blink = ([], [])  
# print(hash(blink)) # так не сработает  
  
# при определении кортежа из одного элемента  
# ставим запятую после первого элемента  
blink = (1, )  
print(hash(blink), type(blink))  
  
# кортежи так же поддерживают срезы  
print(immutable[0: 2])



№10

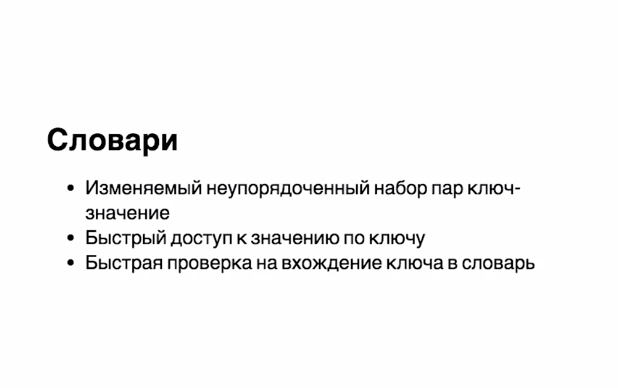
Словари

# словари dict()  
  
# словари хэшируются получаем значения за константное время  
empty\_dict = {}  
empty\_dict = dict()  
  
# если обратиться к словарю по ключу  
# которого не существует, то будет ошибка  
# для этого используем метод get  
testMap = dict([[1, 2], ['a', 3], ['a', 4]])  
print(testMap.get(7, 'not found'))  
  
# в словарь можно добавлять и удалаять элементы  
testMap['test'] = 32 # удаление элемента  
print(testMap)  
  
del testMap['a'] # удаление элмента  
print(testMap)  
  
# удаляем элемент и возвращаем его значение  
print(testMap.pop(1))  
  
  
# для иттерации по словарю  
testMap = dict([['a', 11], ['b', 12], [32, 'ds']])  
  
for elem in testMap.items():  
 print(elem[1])

№ 10.1

OrderedDict

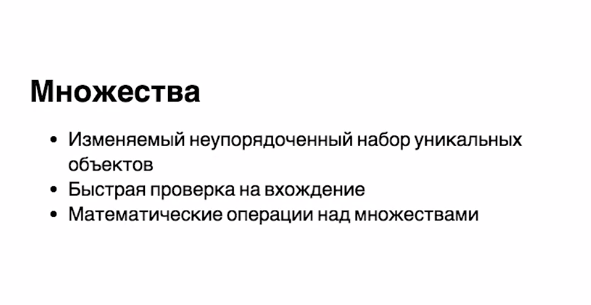
# чтобы словарь мог помнить порядок вхождения элементов  
  
from collections import OrderedDict  
  
ordered = OrderedDict()  
for number in range(10):  
 ordered[number] = str(number)  
  
for key in ordered.keys():  
 print(key)



Урок № 11

Множества

# множества являются хешируемыми  
  
# инициализация множеств  
oddSet = set()  
evenSet = set()  
  
# добавление элемента множества  
oddSet.add(3)  
oddSet.add("3")  
print(oddSet)  
  
# удаление элемента множества  
oddSet.remove(3)  
print(oddSet)  
  
# чтобы сделать множество неизменяемым  
# в данное множество сы не можем добавлятьи удалять элементы  
frozen = frozenset(['Anna', "Rana"])  
  
# frozen.add(3) так не получится сделать



Урок №12

Функции

# функции  
  
# функции могут быть аннотированны  
# передаваемые значения могут несоответствовать типам  
# типы нужны для понимания программистом или IDE  
def add(x: int, y: int) -> int:  
 print(x + y)  
  
def mult(x: [], y: {}) -> None:  
 print(x)  
  
  
# в Python все значения передаются по ссылке  
def multMass(firstTuple: (), secondMass: []) -> None:  
 firstTuple = secondMass  
  
# first = (1, )  
# multMass(first, 3)  
# print(first)  
  
  
# в python есть именованные аргументы  
def say(greeting: str, name: str) -> None:  
 print("{greet} {name} !".format(greet=greeting, name=name))  
  
say(name='Topa',greeting="Opa")

Урок №12.1

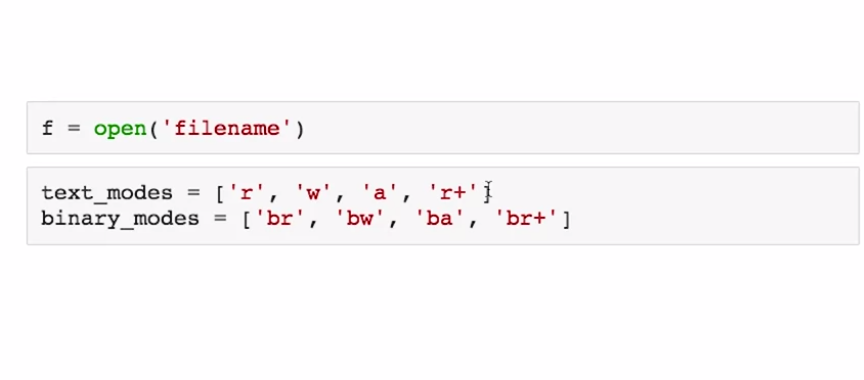
Область видимости

# функции  
  
# аргументы по умолчанию  
def function\_greet(name='unknown'):  
 print(name)  
  
  
function\_greet()  
  
# аргументы по умолчанию для каждой функции  
# интерпретатор записывает в особые переменные  
# поэтому изменяемы типы данных лучше не использовать, как значения по  
# умолчанию, так как они могут быть дозаписаны  
  
def append\_one(arg=[]):  
 arg.append(1)  
 return arg  
  
print(append\_one())  
print(append\_one())  
  
# надо было сделать тип по умолчанию, как None  
def function(iterable=None):  
 iterable = iterable or []  
 return iterable  
  
print(function())  
  
  
# в python функции могут принимать неограниченное  
# кол-во аргументов  
# чтобы задать произвольнео число аргументов  
# используем звёздочку в функции  
# чтобы развернуть произвольное число аргументов  
# так же используем звёздочку  
# при разименовании словаря мы получаем озиционые аргументы  
# и можем передать их в \*args  
def printer(\*args):  
  
 print(args) # кортеж  
 print(\*args) # отдельные элементы  
  
printer(1, 3, 4, 5)  
  
  
def print\_arg(\*\*args):  
 print(args) # словарь ключ-значение  
  
 for elem in args.items(): # получили отдельные значения  
 print("{key}, {value}".format(key=elem[0], value=elem[1]))  
  
print\_arg(a=10, b=12)

# мы можем передавать аргументы в функцию только в определённо  
# порядке, сначала простые аргументы. потом \*args, потом \*kwargs  
# \*args и \*\*kwargs не разрешаются более одного  
def foo(c, d, \*args, \*\*kwargs):  
 print(c, d)  
 print(args)  
 print(kwargs)  
  
  
foo(1, 3, 4, 5, a=10, b=12)

Урок №13

Файлы



# работа с файлами  
  
# файлы можно открывать и записывать в них данные  
file = open('test.txt', 'a+') # режим дозаписи  
for index in range(6):  
 file.write(f"It is str {index}\n")  
  
file.close()  
  
# так же из файлов можно читать данные  
file = open('test.txt', 'r')  
for elem in file:  
 print(elem)  
  
# так же можно узнать где ы нахолимся после прочтения файла  
print(file.tell())  
  
# так же можно переносить указатель на любое место в файле  
file.seek(10) # перенесли указатель на 10 элемент  
print(file.tell())  
file.close()

13.1 Продолжение

# для чтения из файла, так же применяются  
# команды readline и readlines  
# readline считывает строку до символа ереноса строки  
# readlines считывает все строки в массив строк  
  
file = open('test.txt', 'r')  
for elem in file.readlines():  
 print(elem)  
  
file.close()  
  
# для удобной работы с файлами используем  
with open('test.txt', 'r') as f:  
 for line in f: # считываем файл построчно можно было указать f.readlines()  
 print(line)

Урок № 14

Функциональное программирование

# функциональное программирование  
  
  
# map -> проходится по иттерабельному объекту  
# каждый элемент проходит через опрделённую ф-ю  
# возвращает не list а map object  
def foo(param: int) -> int:  
 return param \*\* 2  
  
tempList: int = list(map(foo, range(5)))  
print(tempList)  
  
  
# filter проверяет, что значение соответствует условию  
# условие можно задать функцией  
# из функции мы проверяем, что вернулось значение True или False  
# или знаение .которое можно скастовать к True или False  
# на основании этого происходит запись в list подобный объект  
def is\_positive(a: int) -> bool:  
 if a > 0:  
 return a  
  
list(filter(is\_positive, range(-2, 3)))  
print(tempList)  
  
  
# анонимные функции  
# lambda функции работают внутри другой функции или метода  
# анонимные комбинируются с другими ф-ми типа map, filter  
tempList: [] = list(map(lambda x: x \*\* 4, range(5)))  
print(tempList)

# функиця zip  
# позволяет склеивать иттерабельные объекты  
# сгенерируем два объекта и склеим их  
listTest: [] = list(zip(map(lambda x: x \*\* 2, range(5)), range(5) ))  
print(listTest)

Урок №15

Декораторы

# декораторы  
# декораторы можно применять для создания замыкания  
import functools  
  
def log\_file(path):  
 def decorator\_first(func):  
   
 @functools.wraps(func) #используем чтобы оставить старое имя декоратору  
 def wrapper(\*args, \*\*kwargs):  
 result = func(args, \*\*kwargs)  
 return result  
  
 return wrapper  
  
 return decorator\_first  
  
@log\_file('test.txt')  
def summator(arg):  
 return sum(arg)  
  
print(summator(1, 2, 3, 5, 11))

# декораторы можно собирать в цепочки  
  
def bold(func):  
 def wrapped():  
 return "<b>" + func() + "</b>"  
 return wrapped  
  
def italic(func):  
 def wrapped():  
 return "<i>" + func() + "</i>"  
 return wrapped  
@bold  
@italic  
def hello():  
 return "hello world"  
  
  
# hello = bold(italic(hello)) # как это делалось бы вручную  
print(hello())

Урок №16

Генераторы

# рассмотрим генераторы  
  
# простейший генератор возвращает значения из функции  
# приостанавливая её выполнение  
def digit\_acc(elem: int) -> int:  
 stop = 0  
 while elem > stop:  
 yield stop  
 stop += 2  
  
  
for number in digit\_acc(10):  
 print(number)  
  
# для генератора можно вызвать методы next  
# генератору внутрь ф-ии можно передать значение  
# так же есть метод send для возвращения значения в функцию  
def accumulator():  
 total = 0  
 while True:  
 value = yield total  
 print('Got: {}'.format(value))  
  
 if not value: break  
 total += value  
  
generator = accumulator()  
next(generator)  
generator.send(1)  
generator.send(1)  
generator.send(2)

Урок №17

Классы и экземпляры класса

# классы и объекты  
  
# чтобы проверить относится ли объект к классу  
# используем isinstance(объект, класс)  
string\_type = 'good'  
bool = isinstance(string\_type, str)  
  
  
# объявление класса  
class Human:  
 pass  
  
class Robot:  
 *"""  
 Данный класс определяет робота  
 """*# чтобы узнать методы класса  
# используем dir  
print(dir(Robot))

17.1

# экземпляры класса хешируются и могут быть ключами словаря  
  
class Planet:  
 def \_\_init\_\_(self, name): # инициализация класса  
 self.name = name  
  
 # для внешнего переопределения  
 # может бытть заменён методом \_\_repr\_\_  
 # имеет приоритет над \_\_repr\_\_ во внешнем переопределении  
 def \_\_str\_\_(self):  
 return self.name  
  
 # для внутреннего переопределения имени объекта  
 # является более общим, чем \_\_str\_\_  
 def \_\_repr\_\_(self):  
 return self.name  
  
earth = Planet('Earth')  
print(earth)  
  
  
solar\_system = []  
planet\_name = ['Mercury', "Venus", "Mars"]  
for name in planet\_name:  
 planet = Planet(name)  
 solar\_system.append(planet)  
  
print(solar\_system)  
  
# работа с атрибутами класса  
# мы не можем использовать несуществующие атрибуты  
# объекта, но можем их создавать в коде  
# при удалении несуществующего атрибута выдастся ошибка  
earth.name = "unknown"  
print(earth.name)  
  
earth.test\_attr = "test"  
print(earth.test\_attr)

Урок №18

Классы и экземпляры (часть №2)

# у класса Planet может быть переменная класса  
# к ней можно обратиться через экземпляр и через сам класс  
class Planet:  
  
 # это является \_\_doc\_\_ строкой  
 *"""  
 It is planet class  
 """* count = 0 # переменная класса  
   
 # перед инициализацией \_\_init\_\_ вызывается метод   
 # \_\_new\_\_ в который передаётся класс и с помощью  
 # родительского класса создаётся заготовка объекта  
 # которая будет инициализирована в \_\_init\_\_  
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 print("\_\_new\_\_ called")  
 obj = super().\_\_new\_\_(cls)  
 return obj  
  
 def \_\_init\_\_(self, name, population=None):  
 self.name = name  
 self.population = population or []  
 Planet.count += 1  
  
# можно так же переопределить деструктор  
# когда объект будет удалён будет вызван код внутри деструктора  
class Human:  
 def \_\_del\_\_(self):  
 print("Bye!!!")  
  
human = Human()  
del human  
  
# словарь экземпляра  
# в словаре хранятся все аттрибуты  
# при добавлении новых, они так же добавятся в dict  
planet = Planet('Earth')  
print(planet.\_\_dict\_\_)  
  
planet.age = 12e50  
print(planet.\_\_dict\_\_) # показывает все аттрибуты класса (его переменные)  
print(dir(planet)) # показывает все методы класса (наследованные и самого класса)  
  
print(Planet.\_\_doc\_\_)  
print(planet.\_\_doc\_\_)

Урок №19

# в один класс можно передавать другие объекты  
# из одного метода можно вызвать другой метод  
# методы, который начина.тся с подчёркивания лучше не вызывать  
class Human:  
 def \_\_init\_\_(self, name, age=0):  
 self.name = name  
 self.age = age  
  
  
class Planet:  
 count\_population = 0  
  
 def \_\_init\_\_(self, name, population=None):  
 self.name = name  
 self.population = population or []  
  
 def add\_human(self, human):  
  
 print(f"Welcome to {self.name}, {human.name}")  
 self.population.append(human)  
 self.count\_population += 1  
 self.get\_count\_people()  
  
 def get\_count\_people(self):  
 print(f"Count people: {self.count\_population}")  
  
 def \_say\_anything(self):  
 print("How are you?")  
  
mars = Planet('Mars')  
bob = Human("Bob")  
  
mars.add\_human(bob)  
mars.\_say\_anything() # метод с нижним подчёркиванием лучше не вызывать

№19.1

@Classmethod

# рассмотрим использование @classmethod  
# данный метод позволяет внутри класса определить  
# или инициализировать объект класса  
  
def description\_event():  
 return "Football"  
  
  
def events():  
 return "2018.12.12"  
  
  
class Event:  
  
 countUser = 12  
  
 def \_\_init\_\_(self, description, event\_date):  
 self.desc = description  
 self.event = event\_date  
  
 def \_\_repr\_\_(self):  
 return f"Event: {self.desc} at {self.event}"  
  
 @classmethod  
 def from\_string(cls, user\_input):  
 description = description\_event()  
 event = events()  
 print(f"It is: {cls.countUser}")  
 return cls(description, event)  
  
event = Event.from\_string("Football")

Урок № 20

@staticmethod

# рассмотрим статический метод  
# он не зависит от объекта и может быть вызван по имени класса  
# по имени объекта так же можно вызвать статический метод  
class Human:  
  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 @staticmethod  
 def is\_age\_valid(age):  
 return 0 < age < 150

№20.1

@property

# @property позволяют изменять значения  
# при обращении к ним без рефакторинга кода  
  
class Robot:  
 def \_\_init\_\_(self, name, power):  
 self.name = name  
 self.\_power = power  
  
 power = property()  
  
 @power.setter  
 def power(self, value):  
 if value < 0:  
 self.\_power = 0  
 else:  
 self.\_power = value  
  
 @power.getter  
 def power(self):  
 return self.\_power  
  
 @power.deleter  
 def power(self):  
 print("robot is useless")  
 del self.\_power  
  
robot = Robot('zver', 120)  
robot.power = -12  
print(robot.power)

Урок №21

Наследование

# наследование в python  
# в python наследование может быть множественным  
  
# основной класс  
class Animal(object):  
 count\_animal = 0  
  
 def \_\_init\_\_(self, name):  
 self.name = name  
 self.count\_animal += 1  
  
 def alive\_bool(self):  
 return "Is Alive"  
  
  
# класс примесь  
class EatMeat(object):  
  
 def \_\_init\_\_(self):  
 pass  
  
 def eat\_meat(self):  
 return "i eat meat"  
  
  
# класс наследник  
# вызываем класс родитель, без аргументов родителем считается  
# более левый класс  
class Lion(Animal, EatMeat):  
  
 def \_\_init\_\_(self, name, breed):  
 super().\_\_init\_\_(name)  
 self.breed = breed  
 self.\_\_eat\_now = False # особая приватная переменная  
  
 def roar(self):  
 return "Big roar"  
  
 def is\_eat(self):  
 return self.\_\_eat\_now  
  
lion = Lion('Topa', "Gopa")  
  
# для проверки, что один класс является подклассом другого класса  
# оба аргумента являются классами  
print(issubclass(Lion, Animal))  
  
# для проверки принадлежности переменной к классу  
# первый аргумент переменная  
# второй аргумент класс  
print(isinstance(lion, int))  
  
# для вызова приватный переменных можно использовать  
# особый синтаксис или написать метод для возвращения их объекта  
print(lion.\_Lion\_\_eat\_now)  
print(lion.is\_eat())

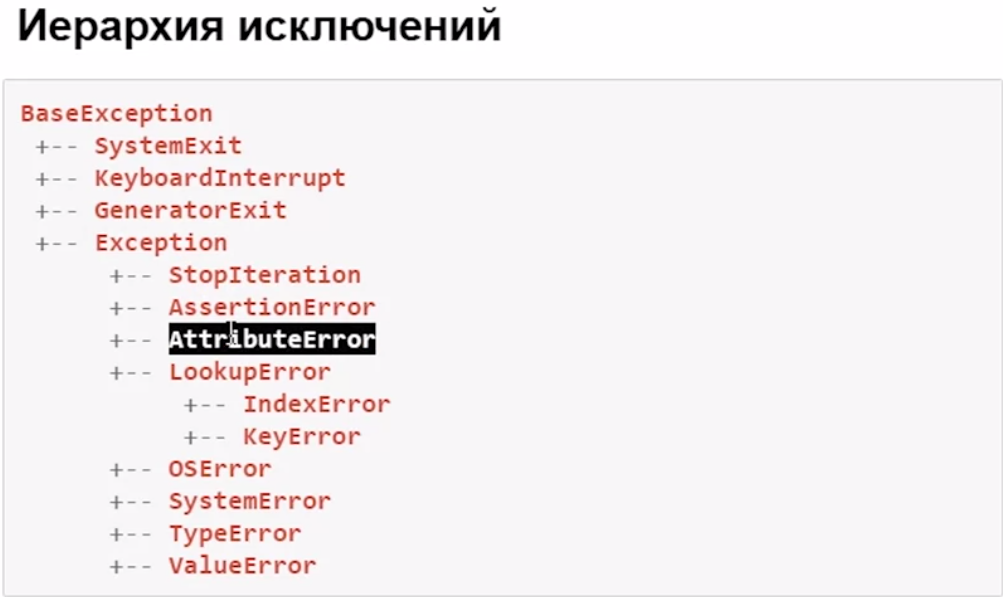
№ 21.1

Композиция

1. Композиция используется, когда очень много классов примесей, мы в одном классе создаём и вызываем объект другого класса (класса примеси)

Урок №22

Исключения python



# рассмотрим исключения на python  
# исключения бывают из стандартной библиотеки  
# пользовательские исключения  
# исключение AttributeError выбрасывается, если в нашем классе нет  
# нужных нам аттрибутов  
  
  
# используем для отлова исключений  
# в except можно указать класс родителя, исключения  
# которые будут отловлены  
# можно использовать блок else  
# он будет выполнен если исключения не произошло  
# можно обрабатывать последовательно несколько  
# блоков исключения  
# можно вносить несколько типов исключений в один exception  
# можно использовать finally, блок который будет выполнен всегда  
try:  
 print(1 / 0)  
except ArithmeticError as e:  
 print(e)  
except (ValueError, KeyError) as ex:  
 print("Global error")  
else:  
 print("No error")  
finally:  
 print("block is closed")  
  
parent\_error = issubclass(ArithmeticError, Exception)  
print(parent\_error)

Урок №23

Генерация исключений

# рассмотрим исключения  
# спомощью ключевого слова raise мы  
# можем выбрасывать исключения  
# в скобках можем указать аргументы, чтобы потом их вызвать  
  
try:  
 print(1 / 10)  
 raise ArithmeticError("Деление на ноль", "Плохо", "Очень плохо")  
except ArithmeticError as err:  
 print(err.args[0], err.args[1], err.args[2], sep="\n")

23.1

# данным способом можно сделать проброс исключений  
# это первый способ  
  
  
def first(arg: int) -> int:  
 try:  
 second(arg)  
 except ArithmeticError as err:  
 print(err.args[0])  
  
  
def second(arg: int) -> int:  
 raise ArithmeticError('Деление на ноль')  
first(56)  
  
  
# второй способ  
# пробрасываем исключение вверх по стеку  
# можно использовать только raise  
# а можно сгенерировать ещё одно исключение и пробросить его по стеку  
# тогда мы увидим всю цепочку исключений  
def test1(arg: int) -> int:  
 try:  
 print(1 / 0)  
 except ArithmeticError as err:  
 print('Производим проброс')  
 raise TypeError('Ошибка в распознование типа') from err  
  
  
def test2(arg: int) -> int:  
 try:  
 test1(arg)  
 except TypeError as val:  
 print(val.args[0])  
test2(45)

23.2

# рассмотрим инструкицю assert  
# данное искючение выбрасывается, если  
# выражение после него ложно, вторым параметром  
# можно передать какое-то значение, которое будет выведено  
# если исключение сработало  
# такие исключения не принято оборачивать в try except  
# так как они предназначены для программистов  
# если запустить программу с флагом -O, то assert перестанут действовать  
  
def test3(arg: int):  
 assert arg > 10, 'Аргумент должен быть строго больше 10'  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 test3(2)

Урок №24

Обработка собственных исключений

# рассмотрим обработку исключений вне стандартной библиотеки  
# командой requests.\_\_file\_\_ узнаём где находится наша библиотека  
# в каждой библиотеке есть класс для обработки исключений  
import requests  
import sys  
  
print(requests.\_\_file\_\_) # узнаём где лежиат файлы нашей библиотеки  
  
url = "https://github.com"  
  
try:  
 response = requests.get(url, timeout=30)  
except requests.Timeout:  
 print('error timeout', url)  
else:  
 print(response.content)  
  
raise None