Тема 2.
Функции, модули, файлы, обработка исключений в Python

Продолжение

Функции в Python

- **Анонимная функция (лямбда-выражение)**
- **В Python лямбда-выражение позволяет создавать анонимные функции – функции, которые не привязаны к имени.**
- **В анонимной функции:**
  - **может содержаться только одно выражение**
  - **могут передаваться сколько угодно аргументов**
  - **не требуют оператора return для возврата результата.**

Функции в Python

- **Анонимная функция (лямбда-выражение)**
- **Схема описания анонимной функции:**

```
lambda [<Параметр1>[, ... , <ПараметрN>]]: <Возвращаемое значение>
```

- **Анонимные функции выполняются быстрее, по сравнению с обычными.**
- **В качестве значения анонимная функция возвращает ссылку на объект-функцию, которую можно сохранить в переменной или передать в качестве параметра в другую функцию.**

Функции в Python

- Стандартная функция

```
def sum_arg(a, b):
    return a + b

print(sum_arg(4, 6))
```

- Анонимная функция (лямбда-выражение)

```
sum_arg2 = lambda a, b: a + b

print(sum_arg2(3, 5))
```

Функции в Python

- **Анонимная функция (лямбда-выражение)**
- **Примеры**

```
f1 = lambda: 10+30
f2 = lambda x, y: (x+y)/2
f3 = lambda a, b, c=3: a+b+c

print(f1())
print(f2(30, 67))
print(f3(20,15))
print(f3(20, 15, 10))
print((lambda x: x//100+x//10%10+x%10)(456))
```

Функции в Python

- **Возврат нескольких значений функциями**
- **В Python поддерживается возврат функциями сразу нескольких значений.**
- **Достаточно перечислить их через запятую после инструкции return.**
- **Типом результата в этом случае будет кортеж (tuple).**
- **Важно, при считывании результата в программе использовать соответствующее количество переменных!**

Функции в Python

- **Возврат нескольких значений функциями**
- **Пример:**

```
def f1(x,y):
    return 3*x, 2-y, x**y # значения функции через запятую

f= lambda x, y: (3*x,2-y, x**y) # значения функции в скобках!

x1,x2,x3=f(20,5) # число переменных равно числу параметров
print(x1, x2, x3)
print(f1(15,6))
print((lambda x, y: (3*x,2-y, x**y))(2,3))
```

Функции в Python

- **Возврат нескольких значений функциями**
- **Пример:**

```
def f1(x,y):
    return 3*x, 2-y, x**y # зн

f= lambda x, y: (3*x,2-y, x*

x1,x2,x3=f(20,5) # число переменных равно числу параметров
print(x1, x2, x3)
print(f1(15,6))
print((lambda x, y: (3*x,2-y, x**y))(2,3))
```

Результат работы кода

```
60 -3 3200000
(45, -4, 11390625)
(6, -1, 8)
```

Функции в Python

- **Функции-генераторы**
- **Функцией-генератором называется функция, которая может возвращать одно значение из нескольких на каждой итерации.**
- **Приостановить выполнение функции и превратить функцию в генератор позволяет ключевое слово yield.**
- **Фактически, yield выдает текущее значение функции, а не возвращает его**

Функции в Python

- **Функции-генераторы**

```
def func(x, y):
    for i in range(1, x+1):
        yield i ** y

for n in func(10, 2):
    print(n, end=" ")       # Выведет: 1 4 9 16 25 36 49 64 81 100
print ()

for n in func(10, 3):
    print(n, end=" ") # Выведет: 1 8 27 64 125 216 343 512 729 1000
```

Функции в Python

- **Функции-генераторы**
- **В сравнении с обычными функциями, возвращающими набор значений в виде списка, функции-генераторы позволяют получать значения из такого набора по одному (т.е. не требуется загружать весь набор в память!).**
- **Функции-генераторы неявно вызывают метод __next__(), для получения следующего значения из набора.**

Функции в Python

- **Функции-генераторы**
- **В Python 3.3 появилась возможность вызывать одну функцию-генератор из другой.**
- **Для этого применяется расширенный синтаксис yield.**

```
yield from <вызываемая функция-генератор>
```

Функции в Python

- **Функции-генераторы**

```
def gen2 (n) :
    for e in range(1, n + 1):
        yield e * 2


def gen (lst):
    for e in lst:
        yield from gen2(e)

mylst=[5, 10]
for i in gen ( mylst ) :
    print(i, end = " ")
```

Результат

2 4 6 8 10 2 4 6 8 10 12 14 16 18 20

Функции в Python

- **Функции-генераторы**

```
import math
def prime(n):
    if n<2 : return False
    for i in range(2, int(math.sqrt(n)+1)):
        if n%i==0: return False
    return True

mylst1 = [1, 2, 3]

def gen1(lst):
    for i in lst:
        yield from gen2(range(10**(i-1), 10**i))
```

```
def gen2(lst):
    f=0
    for i in lst:
        p=prime(i)
        if p: f=1;  yield i
    if f==0: yield 0


for i in gen1(mylst1):
    if i==0:
        print("not found")
    else:
        print(i)
```

Функции в Python

- **Декораторы функций**
- **В Python декоратор – это реализация шаблона, который позволяет добавить поведение к функции или классу.**
- **Обычно это выражается синтаксисом @decorator с префиксом функции**

Функции в Python

- **Декораторы функций**

```
x,y,z=input("input x,y,z: ").split()
x=int(x);y=int(y);z=int(z)

def some_decorator(f):
    def wraps(*args):
        print(f"Вызов функции '{f.__name__}'")
        return f(*args)
    return wraps
```

Результат работы

```
input x,y,z: 23 56 8
Вызов функции 'decorated_function'
С параметрами (23, 56, 8)
```

```
@some_decorator
def decorated_function(x,y,z):
    return f"С параметрами {x,y,z}"

print(decorated_function(x,y,z))
```

Модули

Инструкции import и from. Создание и использование собственных модулей. Обзор стандартной библиотеки Python.

Модули

- **Модуль в Python – это любой файл с программой.**
- **Для подключения модуля к программе его нужно импортировать.**
- **Одной инструкцией можно подключить несколько модулей (не рекомендуется)**
- **Python помещает код импортируемых модулей вместе с остальной частью приложения в память, как будто был создан один огромный файл.**

Модули

Способы импортирования модулей:

- инструкция import – для импорта целого модуля.
  - `import module1[, module2, …]`
  - `import module as new_name`
- Доступ к импортированному содержимому осуществляется через точку:

`имя_модуля.функция`

Модули

Способы импортирования модулей:

- **инструкция import – для импорта целого модуля.**
- **Пример**

```
import math, time
import random as rnd
```

или

```
import math, time, random as rnd
```

```
print(f"time = {time.time()} \t PI = {math.pi}")
print(f"rnd_val = {rnd.random()}")
```

Модули

Способы импортирования модулей:

- **инструкция import – для импорта целого модуля.**
- **Пример**

```
import math, time
import random as
```

Результат:

time = 1601891148.131
PI = 3.141592653589793
rnd_val = 0.043863708333946

```
print(f"time = {time.time()} \t PI = {math.pi}")
print(f"rnd_val = {rnd.random()}")
```

Модули

Способы импортирования модулей:

- **инструкция from … import – импорт отдельного содержимого модуля (например, некоторых (перечисленных) функций):**
  - `from module import func1[, func2, …]`
  - `from module import func as new_func_name`
  - `from module import *`
- **Доступ к импортированному содержимому по имени (без указания модуля)**

Модули

Способы импортирования модулей:

- **инструкция from ... import.**
- **Пример**

```
from math import log, sqrt as sq, cos, ceil


y = ceil(log(20)*sq(42)-cos(30))
print(f"y = {y}")
```

Модули

Функция dir([object_name]):

- **Для просмотра перечня функций, входящих в импортируемый модуль, можно воспользоваться функцией dir()**

```
import math
function_list = dir(math)
for name in function_list:
    print(name, end=", " )
```

__doc__, __loader__, __name__, __package__, __spec__, acos, acosh, asin, asinh, atan, atan2, atanh, ceil, copysign, cos, cosh, degrees, e, erf, erfc, exp, expm1, fabs, factorial, floor, fmod, frexp, fsum, gamma, gcd, hypot, inf, isclose, isfinite, isinf, isnan, ldexp, lgamma, log, log10, log1p, log2, modf, nan, pi, pow, radians, remainder, sin, sinh, sqrt, tan, tanh, tau, trunc

Модули

- **Создание собственного модуля**

1. **Создайте новый файл с расширением .py.**
2. **Добавьте в него атрибуты (определение переменных, функций и др.)**
3. **Импортируйте модуль в своей программе.**

- ***Важно*:**
  - **Расширение файла модуля может быть любым, но импортировать можно только файлы с расширением .py!**

Модули - Пример

Файл my_module.py

```
from math import sqrt

def is_prime(x):
    if x < 2 or x > 2 and x % 2 == 0: return False
    if x == 2: return True
    for i in range(3, int(sqrt(x) + 1), 2):
        if x % i == 0: return False
    return True

def is_perfect(x):
    return sum_del(x)+1 == x

def sum_del(x):
    sum = 0
    for i in range(2, x // 2 + 1):
        if x % i == 0:
            sum += i
    return sum

def check_n():
    n = int(input("input n: "))
    print(f"{n} is prime") if (is_prime(n)) else print(f"{n} is not prime")
    print(f"{n} is perfect") if (is_perfect(n)) else print(f"{n} is not perfect")
```

Файл main.py

```
import my_module
from my_module import sum_del
while True:
    print("1. Check: prime or perfect \n"
          "2. Count sum_delit \n"
          "3. Exit")
    c = int(input("Your choice (1-3)?"))
    if c==1: my_module.check_n()
    elif c==2:
        x = int(input("input x: "))
        print(f"sum_del({x})={sum_del(x)}")
    else : exit(0)
```

Модули

- Создание собственного модуля
- Правила именования модулей:
  - Нельзя использовать в качестве имени зарезервированные ключевые слова или имена встроенных функций!
  - Имя модуля не может начинаться с цифры

Модули

- Создание собственного модуля
- Пути поиска модулей указаны в переменной sys.path
- sys.path по умолчанию включает:
  - Текущую директорию
  - Директории, в которых установлен python
- sys.path можно изменять вручную

```
import sys
print(sys.path)
```

Модули

- **Создание собственного модуля**

```
import sys
print(sys.path)
```

```
['C:\\Users\\User\\PycharmProjects\\pythonProject',
'C:\\Users\\User\\PycharmProjects\\pythonProject',
'C:\\Users\\User\\PycharmProjects\\pythonProject\\venv\\Scripts\\python37.zip',
'D:\\Python\\DLLs', 'D:\\Python\\lib', 'D:\\Python',
'C:\\Users\\User\\PycharmProjects\\pythonProject\\venv',
'C:\\Users\\User\\PycharmProjects\\pythonProject\\venv\\lib\\site-packages']
```

Модули

- **Особенности импорта**
- **При импортировании модуля его код выполняется полностью!**
- **Если нужно запретить выполнение кода модуля, можно воспользоваться переменной __name__**
- **__name__ =**
  - **“__main__”, если скрипт запущен в качестве главной программы**
  - **имя модуля, в случае его импорта.**

Модули - Пример

Файл my_module.py

```
from math import sqrt

def is_prime(x):
    return sum_del(x) == 0

def is_perfect(x):
    return sum_del(x)+1 == x

def sum_del(x):
    …

def check_n():
    …

if __name__ == "__main__" :
    print("Запуск как главной программы")
elif __name__ == "my_module":
    print("Запуск при импорте")
```

Файл main.py

```
import my_module
from my_module import sum_del
while True:
    print("1. Check: prime or perfect \n"
          "2. Count sum_delit \n"
          "3. Exit")
    c = int(input("Your choice (1-3)?"))
    if c==1: my_module.check_n()
    elif c==2:
        x = int(input("input x: "))
        print(f"sum_del({x})={sum_del(x)}")
    else : exit(0)
```

Модули

Основные модули в Python 3

- **math – функционал для работы с числами**
- **random – функции для генерации случайных чисел, букв, и т.п.**
- **os – функции для работы с ОС**
- **datetime – классы для обработки времени и даты**
- **array – работа с массивами**

Модули

Основные модули в Python 3

- **time** – модуль для работы со временим
- **shutil** – функции для обработки файлов, групп файлов, папок и т.д.
- **unittest** – поддерживает автоматизацию тестов и др.
- **calendar** – функции для работы с календарями
- **pickle** – сериализация / десериализация объектов

Модули

Основные модули в Python 3

- **collections – предоставляет специализированные типы данных, на основе словарей, кортежей, множеств, списков**
- **itertools – сборник полезных итераторов.**
- **sys – обеспечивает доступ к некоторым переменным и функциям, взаимодействующим с интерпретатором python.**
- **fractions – поддержка рациональных чисел**
- **и др.**

Исключения

Обработка исключений. Инструкций try ... except. Получение информации об исключениях. Создание новых исключений.

Исключения

- **Примеры исключительных ситуаций в Python:**
  - **NameError – ошибка имени (например, обращение к переменной, имя которой не определено)**

```
a=10
print(a+c)
```

```
Traceback (most recent call last):
  File
"C:/Users/User/PycharmProjects/pythonProject/defs_and_modu
ls.py", line 11, in <module>
    print(a+c)
NameError: name 'c' is not defined
```

Исключения

- **Примеры исключительных ситуаций в Python:**
  - **ValueError – ошибка значения (например, невозможно выполнить операцию над указанным значением)**

```
st = input()
a=int(st)
print(a)
```

```
wew
Traceback (most recent call last):
  File
"C:/Users/User/PycharmProjects/pythonProject/defs_and_moduls.p
y", line 11, in <module>
    a=int(str)
ValueError: invalid literal for int() with base 10: 'wew'
```

Исключения

- **Примеры исключительных ситуаций в Python:**
  - **TypeError – ошибка типа (например, недопустимая операция для используемых в выражении типов переменных)**

```
a=10
b="231"
print(a+b)
```

```
Traceback (most recent call last):
  File
"C:/Users/User/PycharmProjects/pythonProject/defs_and_moduls.
py", line 12, in <module>
    print(a+b)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Исключения

- **Примеры исключительных ситуаций в Python:**
  - **ZeroDivisionError – ошибка деления на ноль**

```
a=int(input("a="))
b=int(input("b="))
print(a/b)
```

```
a=10
b=0
Traceback (most recent call last):
  File
"C:/Users/User/PycharmProjects/pythonProject/defs_and_m
oduls.py", line 12, in <module>
    print(a/b)
ZeroDivisionError: division by zero
```

Исключения

- **Оператор try-except**

```
try:
    инструкции программы
except [имя исключ1 [as имя]]:
    инструкции, после исключения
…
except имя исключN [as имя]:
    инструкции, после исключенияN
[else:
    инструкции, если исключения не было]
[finally:
    инструкции, обязательные к выполнению]
```

Исключения - Пример

```
try:
    a,b=int(input("a=")), int(input("b="))
    print(a/b)
except ValueError:
    print("ValueError exists")
except ZeroDivisionError as zde:
    print(f"Error exists: {zde}") #информация об ошибке через zde
except:
    print(“Something else happened")
else:
    print(“Everything is OK")
finally:  print(“THE END")
```

Исключения

- Оператор try-except
- Для указания набора исключений, который должен обрабатывать блок except, их необходимо перечислить в круглых скобках через запятую

```
try:
    инструкции программы
except (имя исключ1, имя_исключ2, …):
    инструкции, после исключения
```

Исключения

- **Оператор try-except**
  - ***Пример*. Обработка набора исключений**

```
print("start")
try:
    val = int(input("input number: "))
    tmp = 10 / val
    print(tmp)
except (ValueError, ZeroDivisionError):
    print("Error!")
print(“the End")
```

Исключения

- Оператор try-except
- Допустимо вложение операторов try-except

```
try:
    инструкции программы
except (имя исключ1[, имя_исключ2, …]):
    инструкции, после исключения
    try:
        инструкции программы
    except [имя исключ1[, имя_исключ2, …]]:
        инструкции, после исключения
```

Исключения

- Оператор try-except
- Генерация исключений в Python
  - Для принудительной генерации исключений используется инструкция raise

```
try:
    raise Exception("Some exception")
except Exception as e:
    print("Exception exception " + str(e))
```

Исключения

- Иерархия исключений:
- BaseException - базовое исключение, от которого берут начало все остальные
  - SystemExit – исключение, порождаемое функцией sys.exit при выходе из программы
  - KeyboardInterrupt – порождается при прерывании программы пользователем (Ctrl+C)
  - GenerationExit – при вызове close объекта generator.

Исключения

- Иерархия исключений:
- BaseException - базовое исключение, от которого берут начало все остальные
  - Exception – обычные исключения
    - ArithmeticError (FloatingPointError, OverflowError, ZeroDivisionError)
    - EOFError
    - ImportError
    - MemoryError

Исключения

- Иерархия исключений:
  - Exception – обычные исключения
    - NameError
    - OSError
      - FileExistsError
      - NotADirectoryError
      - PermissionError
    - RuntimeError

Файлы

Методы для работы с текстовыми файлами. Функции для работы с файлами. Перенаправление ввод/вывода. Сохранение объектов в файл.

Файлы

- В Python существует два типа файлов:
  - текстовые
  - бинарные.
- Базовые операции над файлами:
  - Открытие / Закрытие
  - Чтение / Запись

Файлы

- **Открытие файла - Метод open()**
  - **Синтаксис**

```
f = open(file_name[, access_mode])
```

  - **где**
    - **file_name – имя открываемого файла**
    - **access_mode – режим открытия (*по умолчанию r*)**
      - **возможные значения: r, r+, w, w+, a, a+, x, x+**
      - **После режима можно использовать модификатор b или *t* (*по умолчанию*): rb, rb+, wb, wb+, ab, ab+, xb, xb+**

Файлы

- **Открытие файла - Метод open()**
  - **По умолчанию файл открывается на чтение в текстовом режиме**
  - **Пример:**

```
f  = open("files/1.txt",encoding="utf-8")
f1 = open("files/1.txt", "rb")
f2 = open("files/3.txt", "w+")
```

Файлы

- **Открытие файла - Метод open()**
  - **Вывод информации о файловом объекте f :**
    - **print(f)**
  - **Вывод содержимого файла f:**
    - **print(*f)**

Файлы

- **Открытие файла - Метод open()**
  - **Пример :**

```
f = open("files/1.txt",encoding="utf-8")
print(f"File {f.name} in text mode: ")
print(*f)

f1 = open("files/1.txt", "rb")
print(f"File {f1.name} in binary mode: ")
print(*f1)
```

Файлы

- **Открытие файла - Метод open()**
  - **Пример :**

Результат работы:

File files/1.txt in text mode:

Лена 8935 111 11 22

Вася 8935 123 45 67

File files/1.txt in binary mode:

b'\xd0\x9b\xd0\xb5\xd0\xbd\xd0\xb0 8935 111 11 22\r\n'

b'\xd0\x92\xd0\xb0\xd1\x81\xd1\x8f 8935 123 45 67\r\n‘

Process finished with exit code 0

Файлы

- **Открытие файла - Метод open()**
  - **У файлового объекта есть следующие атрибуты:**
    - **file.closed – True, если файл закрыт**
    - **file.mode – режим доступа к файлу (файл должен быть открыт)**
    - **file.name – имя файла**

Файлы

- **Закрытие файла - Метод close()**
  - **После открытия файла в Python его нужно закрыть.**
  - **Python автоматически закрывает файл, когда объект присваивается другому файлу.**
  - **Пример:**

```
f   = open("files/1.txt",encoding="utf-8")
… # Работа с файлом f
f.close()
```

Файлы

- **Пример:**
  - **Вывести на экран содержимое заданного файла. Если файл не существует, выдать соответствующее сообщение**

```
def task2():
    fname = input("Введите имя файла: ")
    f = None
    try:
        f=open(fname,"r", encoding="utf-8")
        print(*f)
    except FileNotFoundError:
        print("Файл не существует!")
    except:
        print("Ошибка работы с файлом")
    finally:
        if f: f.close()
```

Файлы

- **Чтение данных из файла**
- **Метод read([размер]) – считывает из файла заданное число символов. Если размер не указан, считывается весь файл.**

```
f = open("files/1.txt",encoding="utf-8")
print(f.read(5))
print(f.read())
```

Результат работы

```
Лена
8935 111 11 22
Вася 8935 123 45 67
Петя 8922 123 33 35
Света 8932 541 21 34
Саша 8918 123 44 25
Маша 8902 323 44 35
```

Файлы

- **Чтение данных из файла**
- **Метод readline() – считывает строку из открытого файла**

```
f = open("files/1.txt",encoding="utf-8")
print(f.readline())
```

Результат работы

Лена 8935 111 11 22

Файлы

- **Чтение данных из файла**
- **Метод readlines() – считывает содержимое файла в список строк**
- **Построчное считывание файла:**

```
f = open("files/1.txt")
for line in f.readlines():
    print(line)
```

```
f = open("files/1.txt")
for line in f:
    print(line)
```

Файлы

- **Запись данных в файл**
- **Метод write(строка) – записывает в файл строку. Возвращает число записанных символов.**
- **Пример.**
  - **Записать все простые двузначные числа в новый файл**

Файлы

- **Запись данных в файл**

Пример (программа):

```
from math import sqrt
def prime(n):
    if (n<2 or n>2 and n%2==0) :
        return False
    if n==2: return True
    for i in range(3, int(sqrt(n)+1), 2):
        if n%i==0: return False
    return True
```

```
f = open("files/3.txt","w")
for i in range(10,100):
    if prime(i):
        f.write(str(i)+" ")
f.close()
f = open("files/3.txt","r")
print(f.read())
f.close()
```

Файлы

- **Дополнительные методы для работы с файлами**
- **Метод tell() – возвращает текущую позицию «условного курсора» в файле**

```
f = open("files/1.txt")
print(f.read(5))
print(f.tell())  # текущая позиция = 5
f.close()
```

Файлы

- **Дополнительные методы для работы с файлами**
- **Метод seek(позиция) – переход к указанной позиции в открытом файле**

```
f = open("files/3.txt","w+")
for i in range(10,100):
    if prime(i):
        f.write(str(i)+" ")
f.seek(0)  # перейти в начало файла
print(f.read(8)) # вывести 8 символов из файла
f.close()
```

Файлы

- **Дополнительные методы для работы с файлами**
- **Метод next() – возвращает следующую строку файла**
- **Метод truncate([n]) – уменьшение размера файла до n байт или до текущей позиции**
- **Метод seekable() – True, если файл поддерживает случайный доступ**
- **Метод writeable() – True, если файл поддерживает запись**

Файлы

- **Дополнительные методы для работы с файлами**
- **Метод flush() – принудительная запись данных из буфера на диск**
- **Метод truncate([n]) – уменьшение размера файла до n байт или до текущей позиции**
- **Метод seekable() – True, если указатель файла можно сдвинуть в другую позицию**
- **Метод writeable() – True, если файл поддерживает запись**

Файлы

- **Функции модуля os**
- **Метод os.getcwd() – возвращает текущую рабочую папку**
- **Метод mkdir(folder_name) – создает пустую папку с указанным именем**
- **Метод chdir(folder_name) – изменить текущую папку на указанную**
- **Метод rename(old_name, new_name) – переименовать файл или папку**

Файлы

- **Функции модуля os**
- **Метод os.listdir() – содержимое текущей папки**
- **Метод remove(name) – удалить файл с указанным именем**
- **Метод rmdir(folder_name) – удалить папку**
- **Метод stat(name) – получение информации о файле**
- **и др.**

Файлы

- **Функции модуля os.path**
- **Метод os.path.abspath(относ.путь) – преобразует относительный путь в абсолютный**
- **Метод:**
  - **os.path.isabs(путь) – True, если путь абсолютный**
  - **os.path.isdir(путь),**
  - **os.path.isfile(путь)**
- **Метод os.path.exists(путь) – True, если файл/папка существуют**