**Лабораторная работа № 1.**

**Виды ошибок**

**Цель:** научиться выявлять первичные и вторичные ошибки программного обеспечения, изучить процесс отладки программного обеспечения ручным методом.

**Ход работы:**

1. Ознакомиться с теоретической частью.
2. Выполнить практическое задание.
3. Оформить отчет.

# Теоретическая часть

При возникновении ошибки времени выполнения Python создает (возбуждает) специальный объект - исключение, который позволяет однозначно характеризовать возникшую ошибочную ситуацию. Выбор подходящего исключения происходит из встроенной иерархии классов-исключений (фрагмент):

* **BaseException** (базовое исключение)
  + **SystemExit** (исключение, порождаемое функцией [**sys.exit()**](https://www.yuripetrov.ru/edu/python/ch_11_01.html#sys.exit) при выходе из программы)
  + **KeyboardInterrupt** (прерывании программы пользователем, Ctrl+C)
  + **Exception** (базовое несистемное исключение)
    - **ArithmeticError** (арифметическая ошибка)
      * **FloatingPointError** (неудачное выполнение операции с плавающей запятой)
      * **OverﬂowError** (результат арифметической операции слишком велик для представления)
      * **ZeroDivisionError** (деление на ноль)
    - **LookupError** (некорректный индекс или ключ)
      * **IndexError** (индекс не входит в диапазон элементов)
      * **KeyError** (несуществующий ключ)
    - **MemoryError** (недостаточно памяти)
    - **NameError** (не найдено переменной с таким именем)
    - **OSError** (ошибка, связанная с ОС - есть подклассы, например **FileNotFoundError**)
    - **SyntaxError** (синтаксическая ошибка, включает классы **IndentationError** и **TabError**)
    - **SystemError** (внутренняя ошибка)
    - **TypeError** (операция применена к объекту несоответствующего типа)
    - **ValueError** (аргумент правильного типа, но некорректного значения)

Пример встроенного возбуждения исключения:

**>>>** "я - строка" / 5

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for /: 'str' and 'int'

### [Конструкция](https://www.yuripetrov.ru/edu/python/ch_07_01.html#id89)[try](https://www.yuripetrov.ru/edu/python/ch_07_01.html#id0)

Придерживаясь идеологии «Легче попросить прощения, чем разрешения», Python предусматривает конструкцию [**try**](https://www.yuripetrov.ru/edu/python/ch_07_01.html#id0) для обработки возникающих исключений.

**try**

**except**

**else**

**finally**

**try**: *# (try строго 1)*

try\_ suite *# код, который может выполниться с ошибкой*

**except** exception\_group1 **as** var1: *# (except - 0 (если есть finally) и более)*

except\_suite1 *# код, выполняемый в случае исключения 'exception\_group1'*

... *# ссылка на исключение может быть записана в 'var1'*

**except** exception\_groupN **as** varN:

except\_suiteN *# код, выполняемый в случае исключения 'exception\_groupN'*

... *# except-блоков может быть произвольное кол-во*

**else**: *# (else - 0 или 1)*

else\_suite *# выполняется, если try не завершен преждевременно (например, break)*

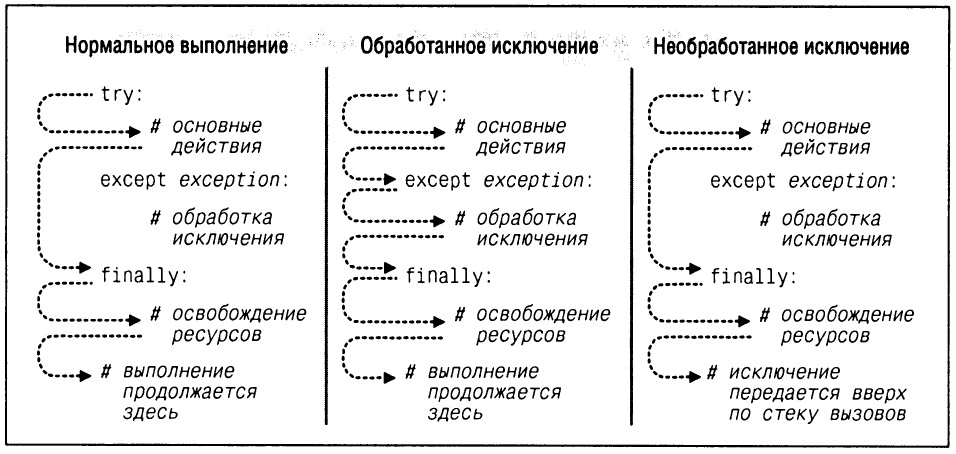
**finally**: *# (finally - 0 или 1)*

finally\_suite *# код, который должен выполнится всегда (была ошибка выше или нет)*

Ход выполнения:

* код, который потенциально может привести к ошибке, помещается в блок [**try**](https://www.yuripetrov.ru/edu/python/ch_07_01.html#id0);
* в случае ошибки, код немедленно завершается и переходит в обработчик [**except**](https://www.yuripetrov.ru/edu/python/ch_07_01.html#except) (если он указан для соответствующего исключения);
* после поток выполнения переходит к [**else**](https://www.yuripetrov.ru/edu/python/ch_07_01.html#else) (если исключений не было) и [**finally**](https://www.yuripetrov.ru/edu/python/ch_07_01.html#finally) (в любом случае).

На Рисунке 1.1 приведены общие варианты потока выполнения программы при обработке исключений.



**Рисунок 1.1 - Варианты потока выполнения программы при обработке исключений**

Обработка исключений (и соответственно идеология «Легче попросить прощения, чем разрешения») - предпочитаемый способ в Python, а использование блоков зависит от конкретной ситуации.

Наиболее общий вариант обработки исключений приведен в Листинге 1.2.

***Листинг 1.2 - Наиболее простой способ обработки исключений |***[скачать](https://www.yuripetrov.ru/edu/python/_downloads/d9cf4777e2fae6dca70af8820f077e27/07_01_03.py)

**try**:

x = int(input("Введите целое число x (для вычисления 1/x): "))

res = 1 / x

print("1/**{}** = **{:.2f}**".format(x, res))

**except**:

print("Произошла ошибка!")

*# --------------*

*# Примеры вывода:*

*# Введите целое число x (для вычисления 1/x): 3*

*# 1/3 = 0.33*

*# Введите целое число x (для вычисления 1/x): qwerty*

*# Произошла ошибка!*

Подобный вариант обработки исключений не рекомендуется, т.к. блок [**except**](https://www.yuripetrov.ru/edu/python/ch_07_01.html#except) будет перехватывать любое исключение, что не позволит точно определить ошибку в коде. Улучшить код можно, добавив обработку исключения по классу (Листинг 7.1.4).

***Листинг 1.3 - Обработка общего класса исключений Exception |***[скачать](https://www.yuripetrov.ru/edu/python/_downloads/2ded59674f50cdb6c3216d9205ae9aa0/07_01_04.py)

**try**:

x = int(input("Введите целое число x (для вычисления 1/x): "))

res = 1 / x

print("1/**{}** = **{:.2f}**".format(x, res))

**except** **Exception** **as** err:

print("Произошла ошибка!")

print("Тип:", type(err))

print("Описание:", err)

*# --------------*

*# Примеры вывода:*

*# Введите целое число x (для вычисления 1/x): 3*

*# 1/3 = 0.33*

*# Введите целое число x (для вычисления 1/x): 5.5*

*# Произошла ошибка!*

*# Тип: <class 'ValueError'>*

*# Описание: invalid literal for int() with base 10: '5.5'*

Рекомендуемым способом обработки исключений является как можно большая конкретизация класса исключения (Листинг 1.4).

***Листинг 1.4 - Обработка конкретных классов исключений |***[скачать](https://www.yuripetrov.ru/edu/python/_downloads/fd5a6660384888f0d1e82b6fbfd81ac6/07_01_05.py)

**try**:

x = int(input("Введите целое число x (для вычисления 1/x): "))

res = 1 / x

print("1/**{}** = **{:.2f}**".format(x, res))

**except** **ZeroDivisionError**:

print("На ноль делить нельзя!")

**except** **ValueError** **as** err: *# 'err' содержит ссылку на исключение*

print("Будьте внимательны:", err)

**except** (**FileExistsError**, **FileNotFoundError**): *# Исключения можно перечислять в виде кортежа*

print("Этого никогда не случится - мы не работаем с файлами")

**except** **Exception** **as** err:

*# Все, что не обработано выше и является потомком 'Exception',*

*# будет обработано здесь*

print("Произошла ошибка!")

print("Тип:", type(err))

print("Описание:", err)

*# --------------*

*# Примеры вывода:*

*# Введите целое число x (для вычисления 1/x): 3*

*# 1/3 = 0.33*

*# Введите целое число x (для вычисления 1/x): 0*

*# На ноль делить нельзя!*

*# Введите целое число x (для вычисления 1/x): qwerty*

*# Будьте внимательны: invalid literal for int() with base 10: 'qwerty'*

# Практическая часть

**Задание.** Выполните ручное тестирование написанной программы (Приложение 1), по условиям заданий ниже. Прокомментируйте каждую строку кода.

Отчет по работе должен содержать:

1. Название работы
2. ФИО, группа студента
3. Номер задания, условия задачи (код программы)
4. Таблица результатов выполнения кода с указанием ошибок.

**Требования к оформлению работы:** шрифт Times New Roman, 12 кегль, формат – по ширине страницы, отступ 1-й строки – 1, 25 см, межстрочный интервал – 1,5.

Оценка за работу снижается на 1 балл за несоответствие данным требованиям.

Работа высылается в электронный журнал преподавателю в день проведения занятия. В теме письма указывается **номер** и **название работы.**

**Для каждой задачи 1-5:**

* используя визуальный просмотр кода, простое журналирование, проверку PEP8 или отладчик в IDE (приложение 1):
  + разберитесь в алгоритме решения;
  + найдите номера строк, в которых есть ошибки, и укажите их в строке документации;
* исправьте ошибки, минимально изменив код (как правило, заменив строчку на строчку); ошибки исправляются по ходу программы, сверху вниз.

Пример.

Исходная программа:

*"""*

*Ошибки (номера строк через пробел): !!!*

*"""*

**def** even\_mult(nums):

*"""Вернуть произведение четных чисел в списке 'nums'."""*

p = 0

**for** item **in** nums:

**if** item % 2 != 0:

p \*= item

**return** p

Исправленная программа:

*"""*

*Ошибки (номера строк через пробел с первой строки): 3 5*

*"""*

**def** even\_mult(nums):

*"""Вернуть произведение четных чисел в списке 'nums'."""*

p = 1

**for** item **in** nums:

**if** item % 2 == 0:

p \*= item

**return** p

**Задача 1.**

def sum\_of\_digits(n):

"""Вернуть сумму цифр меньших 5 для положительного целого числа `n`.

Если таких цифр нет, вернуть 0."""

с = 0

while n > 0:

digit = n // 10

if digit < 5:

c = с + 1

n //= 10

return digit

**Задача 2.**

def primes(a, b):

"""Вернуть список простых чисел на отрезке от 'a' до 'b'."""

res = []

c = 0

for i in range(a, b):

for j in range(i):

if i % (j + 1) == 0:

c += 1

if c == 2:

res.append(i)

return res

**Задача 3.**

def first\_vacant\_row(seats):

"""Вернуть первый ряд, в котором имеется больше всего

свободных мест и их количество.

Возвращаемая нумерация рядов с 1. Если свободных мест нет, вернуть 0, 0.

Параметры:

- seats (list of list): информация о проданных билетах

(1 - продано, 0 - нет).

Результат:

- tuple (ряд, количество мест).

"""

max\_count = 0

max\_row = 0

for row\_index, row in enumerate(seats):

available\_seats\_count = row.count(0) # 0 - пусто

if available\_seats\_count >= max\_count:

max\_row = row\_index

max\_count = available\_seats\_count

return max\_row, max\_count

# import random

#

# random.seed(50)

#

# ROWS\_MAX = 10

# SEATS\_MAX = 5

#

# seats = [[random.randint(0, 1) for seat in range(SEATS\_MAX)]

# for row in range(ROWS\_MAX)]

#

# for seat in seats:

# print(seat)

#

# print(first\_vacant\_row(seats))

**Задача 4.**

def min\_pair(nums):

"""Вернуть минимальную сумму соседних 2-х чисел в списке 'nums'."""

min = nums[0] \* nums[1]

for i in range(2, len(nums)):

min = min(nums[i] + nums[i + 1], min)

return min

# import random

#

# random.seed(50)

#

# N\_MAX = 10

# RANGE\_MIN = 1

# RANGE\_MAX = 100

# nums = random.sample(range(RANGE\_MIN, RANGE\_MAX), N\_MAX)

#

# print(nums)

#

# print(min\_pair(nums))

**Задача 5.**

def non\_negatives(nums):

"""Удалить из списка чисел 'nums' отрицательные элементы и вернуть

измененный список."""

for i in range(len(nums)):

if nums[i] < 0:

del nums[i]

return nums

# import random

#

# n = 10

# nums = [round(random.uniform(-10, 10), 2) for i in range(n)]

# print(nums)

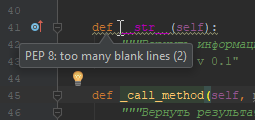
#

# non\_negatives(nums)

# print(nums)

**Приложение 1.**

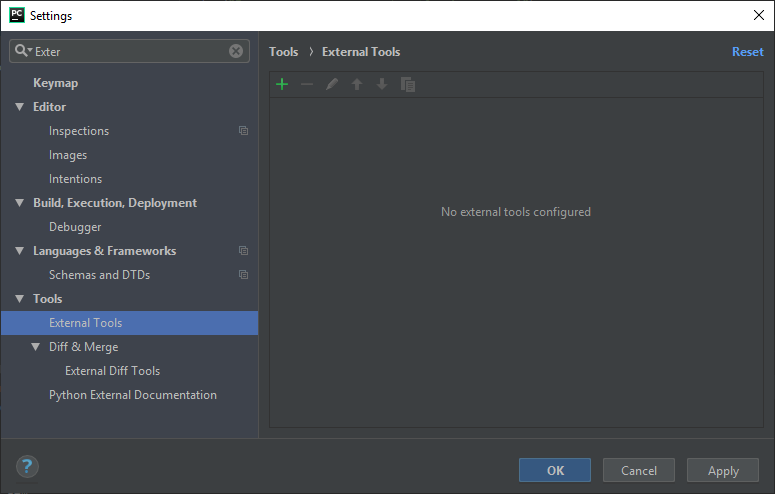
PyCharm автоматически проверяет код на соответствие стандарту [**PEP 8**](https://www.python.org/dev/peps/pep-0008), используя подчеркивание в редакторе кода (Рисунок 1.6).



**Рисунок 1.6 - Автоматическая проверка PEP8 в PyCharm**

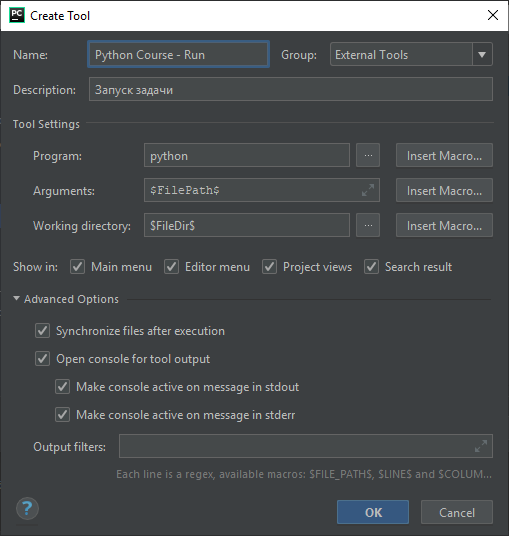
Для быстрого запуска и проверки выполняемых заданий необходимо выполнить ряд настроек.

В главном меню PyCharm выберите File -> Settings и найдите в поиске меню External Tools (Рисунок 1.7).



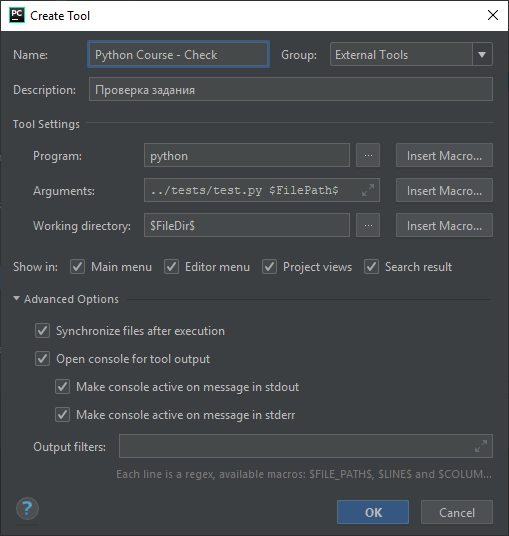
**Рисунок 1.7 - Окно добавления инструментов для запуска и проверки заданий**

Для возможности запуска задания добавьте (через кнопку +) возможность запуска программы (Рисунок 1.8) и нажмите OK.



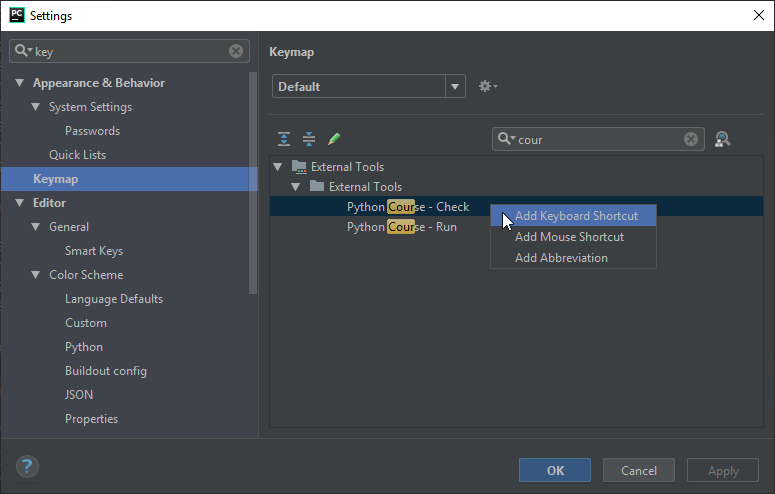
**Рисунок 1.8 - Команда запуска задания**

Аналогично добавьте возможность проверки задания (Рисунок 1.9) и нажмите OK, где путь ..\tests\test.py - относительный путь к валидатору test.py.



**Рисунок 1.9 - Команда проверки задания**

Для быстрого запуска и проверки заданий также рекомендуется добавить горячие клавиши. В главном меню PyCharm выберите File -> Settings и найдите в поиске меню Keymap (Рисунок 1.10).



**Рисунок 1.10 - Окно настройки сочетаний клавиш**

Для добавления сочетания клавиш:

* выделите строку с командой;
* вызовите контекстное меню и выберите пункт Add Keyboard Shortcut;
* в открывшемся окне нажмите горячую клавишу (F5 для запуска, F6 - для проверки) и нажмите OK;
* в случае предупреждения, что горячая клавиша уже занята, нажмите Remove (в дальнейшем сочетания можно будет сбросить к начальным настройкам при необходимости).

#### Сочетания клавиш

Некоторые полезные сочетания клавиш приведены в Таблице 1.1.

| **Действие** | **PyCharm** |
| --- | --- |
| Запуск программы | F5 |
| Запуск автоматической проверки | F6 |
| Комментирование/Раскомментирование строки | Ctrl + / |
| Дублирование текущей строки | Ctrl + D |
| Поиск, замена | Ctrl + F, Ctrl + R |

#### Дополнительные советы

Ряд проблем часто возникает при написании первых программ - используйте советы ниже для их решения.

Использование кириллицы и пробелов

Старайтесь избегать использования кириллицы и пробелов при наименовании имен файлов и идентификаторов в программе.

Кодировка файла

Для вновь создаваемого файла IDE устанавливает кодировку в UTF-8, которая как раз требуется для файлов исходного кода Python, и указывает это в строке состояния. Если файл определяется в другой кодировке (такое бывает, например, если создать файл в блокноте ОС Windows), ее необходимо поменять, выполнив следующие действия:

* Atom: щелкнуть в строке состояния на текущую кодировку файла и выбрать UTF-8,
* Geany: главное меню Документ -> Установить кодировку -> Юникод -> Юникод (UTF-8),
* PyCharm: щелкнуть в строке состояния на текущую кодировку файла и выбрать UTF-8

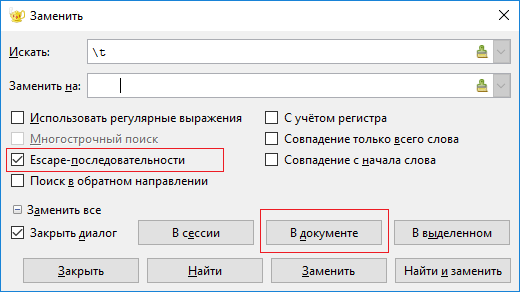
после чего сохранить файл.

Совмещение пробелов и знаков табуляции

Python не позволяет использовать одновременно пробелы и знаки табуляции в качестве разделителей, выдавая ошибку

TabError: inconsistent use of tabs and spaces in indentation.

Данную проблему можно решить, используя замену (Рисунок 1.11).



**Рисунок 1.11 - Замена всех знаков табуляции на 4 пробела на примере Geany**