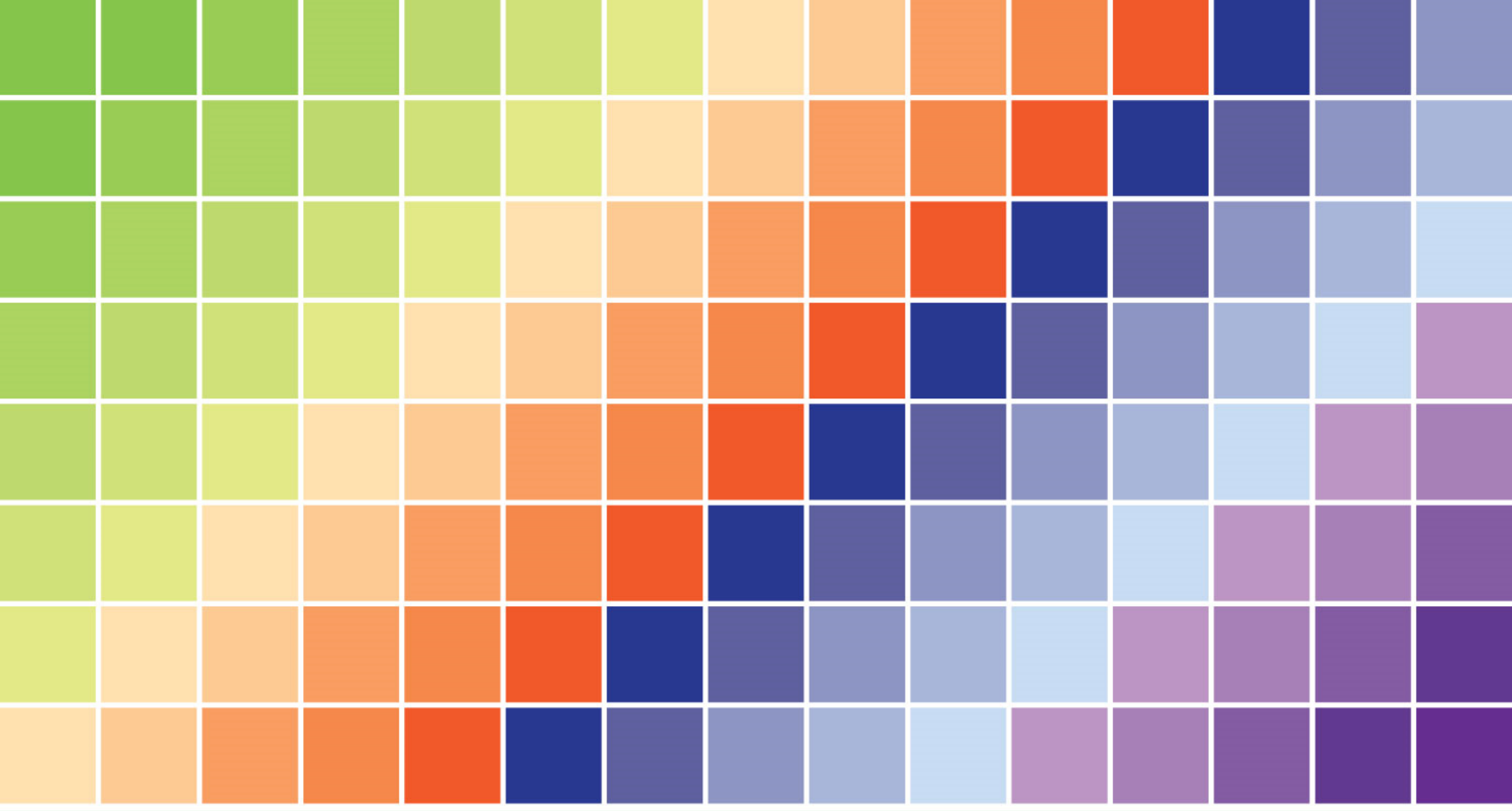


School of Computer Science

ПОВТОРЕНИЕ НА ПОСЛЕДОВАТЕЛЬНОСТИ И ФУНКЦИИ. РЕКУРСИЯ. ИСКЛЮЧЕНИЯ, СЕРИЯ №3

ПРОГРАММИРОВАНИЕ НА PYTHON

Лекции для IT-школы

ВОПРОСЫ ПО ПРОШЛЫМ ЗАНЯТИЯМ. СПИСКИ

- Есть такой список:

  ```
  >>> temperatures = [-3, 0, 2.5, 4]
  ```

- Какое выражение даст значение 4:
  1. `>>> temperatures[-1]`
  2. `>>> temperatures[4]`
  3. `>>> temperatures[3:]`
  4. `>>> temperatures[3:4]`

ВОПРОСЫ ПО ПРОШЛЫМ ЗАНЯТИЯМ. СПИСКИ

– Выберите выражения, которые дадут 4:

1. >>> len([1, 2, 3, 4])
2. >>> len(["math"])
3. >>> min([10, 8, 4])
4. >>> sum([4])

ВОПРОСЫ ПО ПРОШЛЫМ ЗАНЯТИЯМ. СПИСКИ

- Есть такой код:

```
>>> grades = [80, 70, 60, 90]
>>> grades.sort()
>>> grades.insert(1, 95)
```

- На что, в итоге, ссылается grades:
  1. [60, 70, 80, 90, 95]
  2. [95, 60, 70, 80, 90]
  3. [60, 95, 80, 90]
  4. [60, 95, 70, 80, 90]

ВОПРОСЫ ПО ПРОШЛЫМ ЗАНЯТИЯМ. ФУНКЦИИ. МНОЖЕСТВЕННЫЙ ВОЗВРАТ ЗНАЧЕНИЙ

– Как вернуть из функции несколько значений?

1. При помощи возврата кортежа
2. Несколько раз написать return
3. Создать новый пользовательский тип
4. В языке Python это невозможно

ВОПРОСЫ ПО ПРОШЛЫМ ЗАНЯТИЯМ.
ПУСТЫЕ ФУНКЦИИ

- Как создать шаблон функции, но отложить реализацию кода этой функции на будущее?

1. Определить прототип функции, но оставить тело функции пустым
2. Добавить в теле функции строку со служебным словом pass
3. Добавить несколько пустых строк после определения функции

ВОПРОСЫ ПО ПРОШЛЫМ ЗАНЯТИЯМ.
ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ

```
>>> default_var = 1
>>> def sum(param1, param2 = default_var):
        return param1 + param2
>>> default_var = 5
>>> print(sum(6))
```

Каким будет результат выполнения программы?

ВОПРОСЫ ПО ПРОШЛЫМ ЗАНЯТИЯМ. ПЕРЕДАЧА ПАРАМЕТРОВ ПО ССЫЛКЕ

```
>>> animals = [”Cat”, ”Dog”]

>>> def upgrade_animals(p_list):
        p_list = [”Tiger”, ”Wolf”]

>>> upgrade_animals(animals)
>>> animals
```

Каким будет результат выполнения программы и почему?

1. [”Cat”, ”Dog”]
2. [”Tiger”, ”Wolf”]
3. [”Cat”, ”Dog”, ”Tiger”, ”Wolf”]

ВОПРОСЫ ПО ПРОШЛЫМ ЗАНЯТИЯМ.
АНОНИМНЫЕ ФУНКЦИИ

Найдите ошибки в определении анонимной функции:

```
>>> def lambda x, y:
        x *= 2
        return x - y
```

ВОПРОСЫ ПО ПРОШЛЫМ ЗАНЯТИЯМ.
ОБЛАСТИ ВИДИМОСТИ

```
>>> value = 10

>>> def assign_value():
        value = -10

>>> assign_value()
```

В какой области видимости находится переменная *value*?

Чему равно её значение?

РЕКУРСИВНЫЕ ФУНКЦИИ

Рекурсивная функция – функция, которая обращается к самой себе.

Рекурсию используют, когда вычисление функции можно свести к её более простому вызову, а его – к ещё более простому и так далее, пока значение не станет очевидно.

Любая рекурсивная функция может быть переписана в итеративную (с использованием циклов).

РЕКУРСИВНЫЕ ФУНКЦИИ

Каждый рекурсивный вызов называется *шагом рекурсии*.

Общее количество вложенных вызовов (включая первый) называют *глубиной рекурсии*.

База рекурсии – это такие аргументы функции, которые делают задачу настолько простой, что решение не требует дальнейших вложенных вызовов.

КОНТЕКСТ ВЫПОЛНЕНИЯ. СТЕК

Контекст выполнения – специальная внутренняя структура данных, которая содержит информацию о вызове функции. Один вызов функции имеет ровно один контекст выполнения, связанный с ним.

Когда функция производит вложенный вызов, происходит следующее:

- Выполнение текущей функции приостанавливается.
- Контекст выполнения, связанный с ней, запоминается в специальной структуре данных – *стеке контекстов выполнения*.
- Выполняются вложенные вызовы, для каждого из которых создаётся свой контекст выполнения.
- После их завершения старый контекст достаётся из стека, и выполнение внешней функции возобновляется с того места, где она была остановлена.

ПЕРЕД ИСПОЛЬЗОВАНИЕМ РЕКУРСИИ

Для того, чтобы реализовать рекурсию нужно ответить на следующие вопросы:

- Какой случай (для какого набора параметров) будет крайним (простым) и что функция возвращает в этом случае?

- Как свести задачу для какого-то набора параметров (за исключением крайнего случая) к задаче, для другого набора параметров (как правило, с меньшими значениями)?

ИТЕРАТИВНОЕ ВЫЧИСЛЕНИЕ ФАКТОРИАЛА

```
>>> def fact(n):
    if n == 0:
        return 1
    result = 1
    for i in range(1, n+1):
        result *= i

    return result
```

Какой случай является крайним, т.е. что является базой рекурсии?

Как свести задачу к такой же задаче, но с меньшим по значению параметром?

РЕКУРСИВНОЕ ВЫЧИСЛЕНИЕ ФАКТОРИАЛА

```
>>> def factorial(n):
        if n == 0:
            return 1
        else:
            return n * factorial(n - 1)

>>> factorial(10)
120
>>> factorial(2000)
RecursionError: maximum recursion depth exceeded in
comparison
```

Максимальная глубина рекурсии 1000, её можно (но крайне нежелательно!) изменить с помощью функции *sys.setrecursionlimit()*

ПРАКТИЧЕСКОЕ ЗАДАНИЕ
ПРОВЕРКА СТРОКИ НА ПАЛИНДРОМНОСТЬ

Строка является палиндромом, если она одинаково читается как справа налево, так и слева направо. Напишите функцию *IsPalindrome*, которая возвращает значение типа *bool* в зависимости от того, является ли строка палиндромом.

Каким будет крайнее значение?
Как должен выглядеть рекурсивный переход?

ТИПИЧНЫЙ СОСТАВ DOC STRING

1. Типы параметров и возвращаемых значений
2. Описание того, что делает функция
3. Условия ее использования, preconditions
4. Примеры вызовов в стиле Shell

– Смотрите примеры в triangle.py, convert_bin_dec.py, grade_template.py

ТЕСТЫ ФУНКЦИИ ЧЕРЕЗ DOC STRING

- Загрузите в Shell скрипт func_test.py
- Импортируйте модуль doctest:

  import doctest
- Протестируйте функцию:

  doctest.testmod()
- Определите ошибки в тестовых кейсах
- Исправьте тестовые кейсы и снова запустите тестирование

ПОДГОТОВКА HTML-ДОКУМЕНТАЦИИ TRIANGLE.HTML С ПОМОЩЬЮ PYDOC

Стандартный модуль pydoc, функция writedoc() используется для сохранения документации из docstring в HTML-документе

```
>>> import pydoc
>>> import os
>>>
>>> os.getcwd()
'C:\\Python37'
>>> os.chdir(r"C:\IT-School\Python\Year 11 2019-2020\Lesson 7\Scripts\DocstringSamples")
>>> os.getcwd()
'C:\\IT-School\\Python\\Year 11 2019-2020\\Lesson 7\\Scripts\\DocstringSamples'
>>>
>>> import triangle
>>> pydoc.writedoc(triangle)
wrote triangle.html
```

ПРАКТИЧЕСКОЕ ЗАДАНИЕ. ФУНКЦИИ

- Напишите функцию generate_password() по скрипту NonFunc\awful_password.py:
  - Параметр – длина пароля, по умолчанию = 8
  - Глобальные переменные – не используются
  - Функция возвращает сгенерированный пароль
- Напишите функцию calc_latin_letters() по скрипту NonFunc\for_string.py
- Напишите функцию xml_tag_value() по скрипту NonFunc\xml_parser.py
- Протестируйте функции с помощью doctest.testmod() там, где это возможно

ФУНКЦИЯ ENUMERATE()

```
>>> the_list = [10, 20, 30, 40]
>>> for tup in enumerate(the_list):
        print(tup)


(0, 10)
(1, 20)
(2, 30)
(3, 40)
>>> for tup in enumerate(the_list, 1):
        print(tup)


(1, 10)
(2, 20)
(3, 30)
(4, 40)
```

ИСКЛЮЧЕНИЯ. ПРОДОЛЖЕНИЕ

```
try:
    испытуемый код
except (exc1, exc2) [as variable1]:
    реакция на исключения группы 1
except excN [as variableN]:
    реакция на исключения группы N
[else:
    блок, “когда исключений не было”]
[finally:
    блок финальной обработки]
```

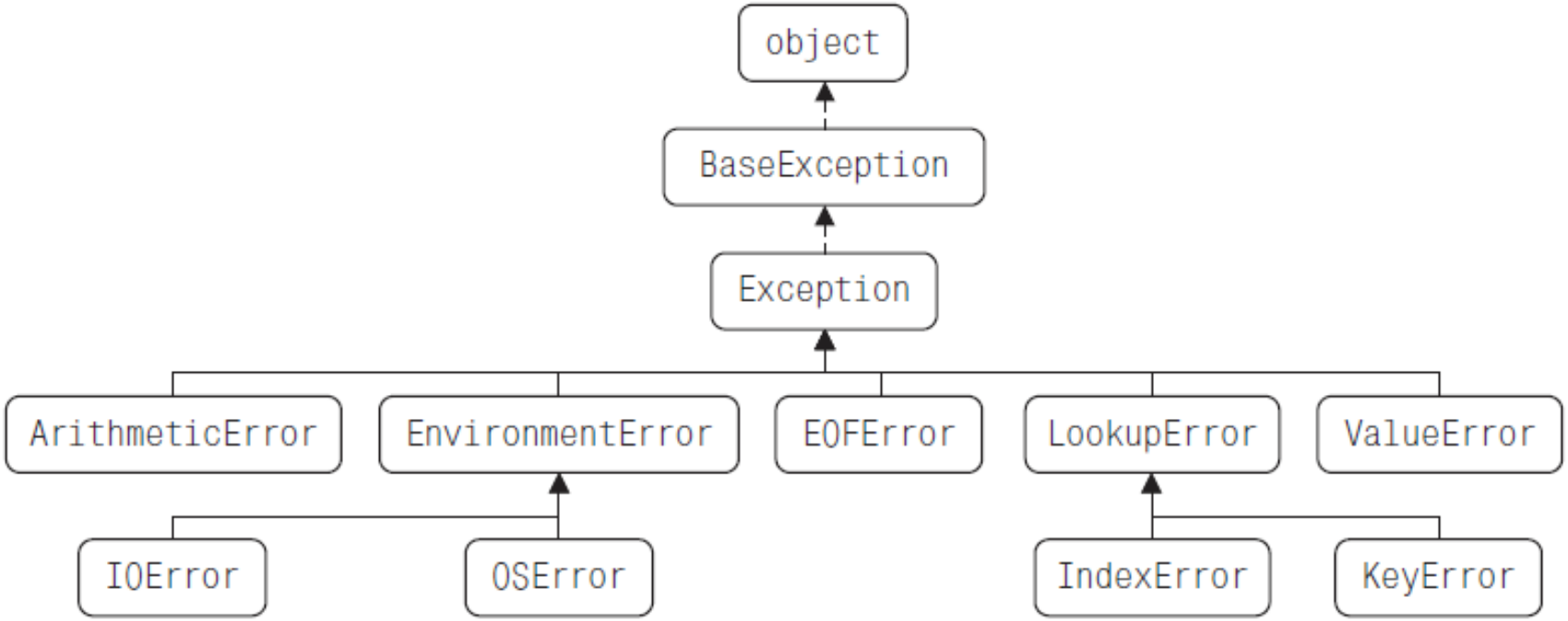
ПОРЯДОК ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Потоки выполнения конструкции *try* … *except* … *finally*

ИСКЛЮЧЕНИЯ.
ПРИМЕР ФИНАЛЬНОЙ ОБРАБОТКИ

- См. скрипт в no_blanks.py
- finally используется для надежной обработки ошибок
- finally вызывается **всегда**, вне зависимости от того было какое-то исключение или нет, предусмотрены для него обработчики или нет

ИЕРАРХИЯ ИСКЛЮЧЕНИЙ



Фрагмент иерархии классов исключений Python

ИЕРАРХИЯ ИСКЛЮЧЕНИЙ. ГДЕ ПОСМОТРЕТЬ?

- Если хотите почитать про системные исключения Python, добро пожаловать:
  - https://docs.python.org/3/library/exceptions.html

- А чтобы увидеть исключения прямо в Python, используйте скрипт print_exceptions.py
- А как вывести результаты исполнения этого скрипта в файл, не меняя саму программу?

ОДИН ОБРАБОТЧИК EXCEPT НА ВСЕ. В ЧЕМ ПРОБЛЕМА?

- Рассмотрите приведенные 2 блока кода
- Какой из них лучше и почему?
- Как можно сделать еще хуже, чем в самом плохом варианте обработки исключения?

Вариант 1:

```
try:
    do_something()
except:
    process_errors()
```

Вариант 2:

```
try:
    do_something()
except Exception:
    process_errors()
```

РОДСТВЕННЫЕ ИСКЛЮЧЕНИЯ. ПРЕДОСТЕРЕЖЕНИЕ

- Несколько блоков except нужно располагать сверху вниз в порядке от более специализированных к более общим

```
>>> test_list = [1, 2, 3]
>>>
try:
    item = test_list[5]
except LookupError: # НЕВЕРНЫЙ ПОРЯДОК ИСКЛЮЧЕНИЙ
    print("Lookup error occurred")
except IndexError:  # ЭТА ЛОВУШКА ДОЛЖНА БЫТЬ ВЫШЕ
    print("Invalid index used")


Lookup error occurred
```

ИСКЛЮЧЕНИЯ.
СОБСТВЕННЫЕ ИСКЛЮЧЕНИЯ

- Пользовательское исключение – это наш собственный тип данных
- Создание своего типа исключения:
  class UserExceptionName(Exception): pass
- Генерация собственного исключения:
  raise UserExceptionName()
- Это используется для:
  - Описания пользовательских типов ошибок
  - Управления потоком выполнения программы

ПРИМЕР КОДА. БЕЗ ИСПОЛЬЗОВАНИЯ ИСКЛЮЧЕНИЙ

```
>>> found = False
>>>
for row, record in enumerate(table):
    for column, field in enumerate(record):
        for index, item in enumerate(field):
            if item == target:
                found = True
                break
        if found:
            break
    if found:
        break
if found:
    print("found at ({0}, {1}, {2})".format(row, column, index))
else:
    print("not found")
```

ОПТИМИЗАЦИЯ КОДА. С ИСПОЛЬЗОВАНИЕМ ИСКЛЮЧЕНИЙ

```
>>> class FoundException(Exception): pass

>>>
try:
    for row, record in enumerate(table):
        for column, field in enumerate(record):
            for index, item in enumerate(field):
                if item == target:
                    raise FoundException()
except FoundException:
    print("found at ({0}, {1}, {2})".format(row, column, index))
else:
    print("not found")
```

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ. ВИКТОРИНА

- План программы:
  - Представить тему и поздороваться
  - Пока не достигли конца файла:
    - Считывать блоки с вопросами
    - Задавать вопросы и запрашивать ответ
    - Проверять ответ на правильность
    - Подсчитывать количество верных ответов
  - Информировать о количестве очков
- См. вопросы викторины в py_struct.txt
- Разработайте программу по шаблону в quiz_template.py

ДОМАШНЕЕ ЗАДАНИЕ. ПРОГРАММА «ВИКТОРИНА» V 2.0

- Добавьте цену (вес) вопроса в структуре файла с данными для указания уровня сложности вопроса
- В конце игры сумма очков должна учитывать вес каждого вопроса
- Пользователь должен указывать свое имя при старте программы
- Добавьте хранение списка рекордов в отдельном файле **в виде словаря (*)**: "<*имя пользователя*>:<*рекорд*>\n"…
- Создайте доп. эпизоды для проверки знаний по работе с исключениями и файлами

СПАСИБО ЗА ВНИМАНИЕ !

ВОПРОСЫ ?

School of Computer Science