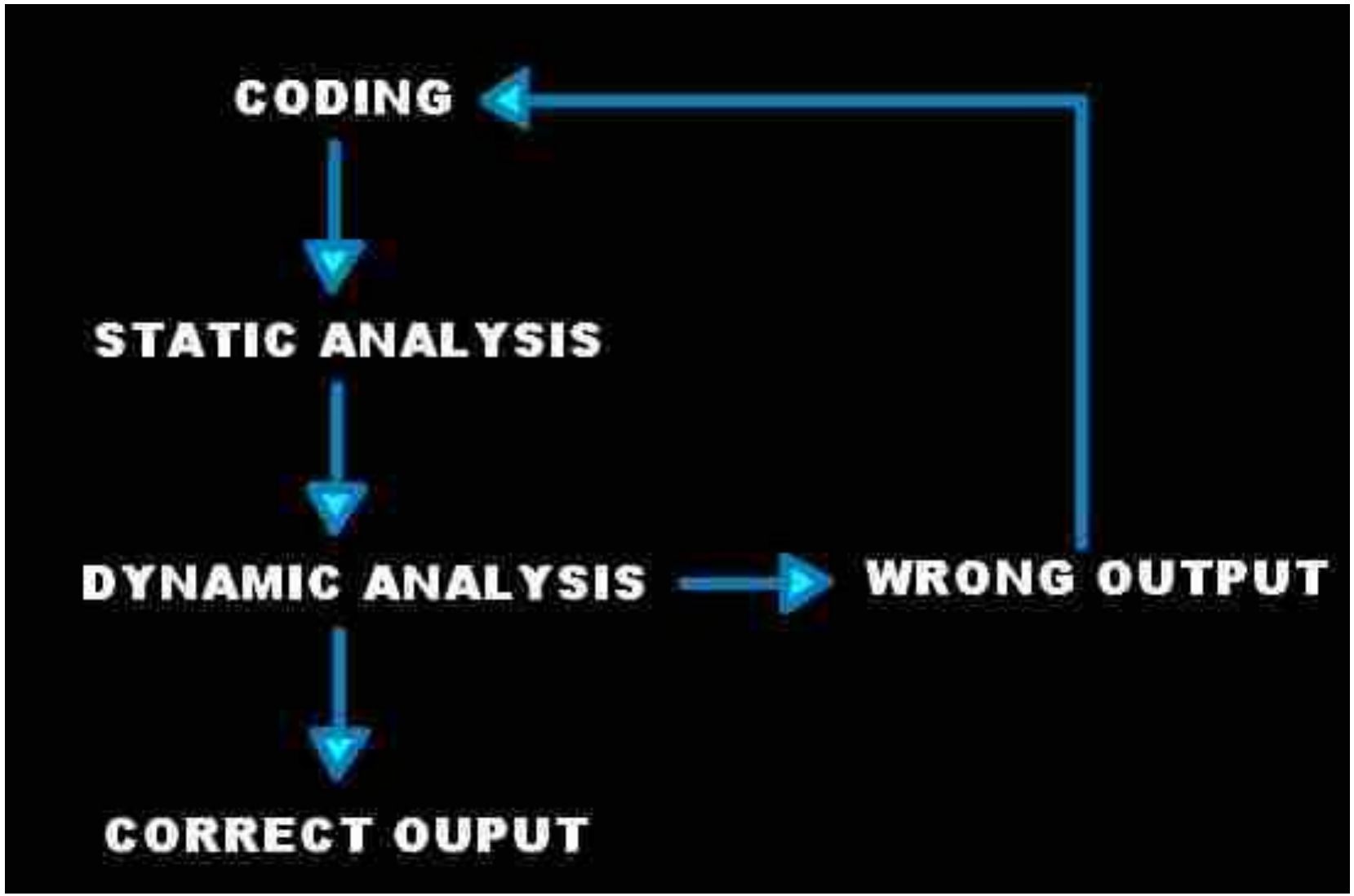
Исключения, ошибки синтаксиса


CODING
STATIC ANALYSIS
DYNAMIC ANALYSIS
WRONG OUTPUT
CORRECT OUPUT

Ошибка синтаксиса

программист нарушает правила, по которым работает парсер языка в интерпретаторе/компиляторе

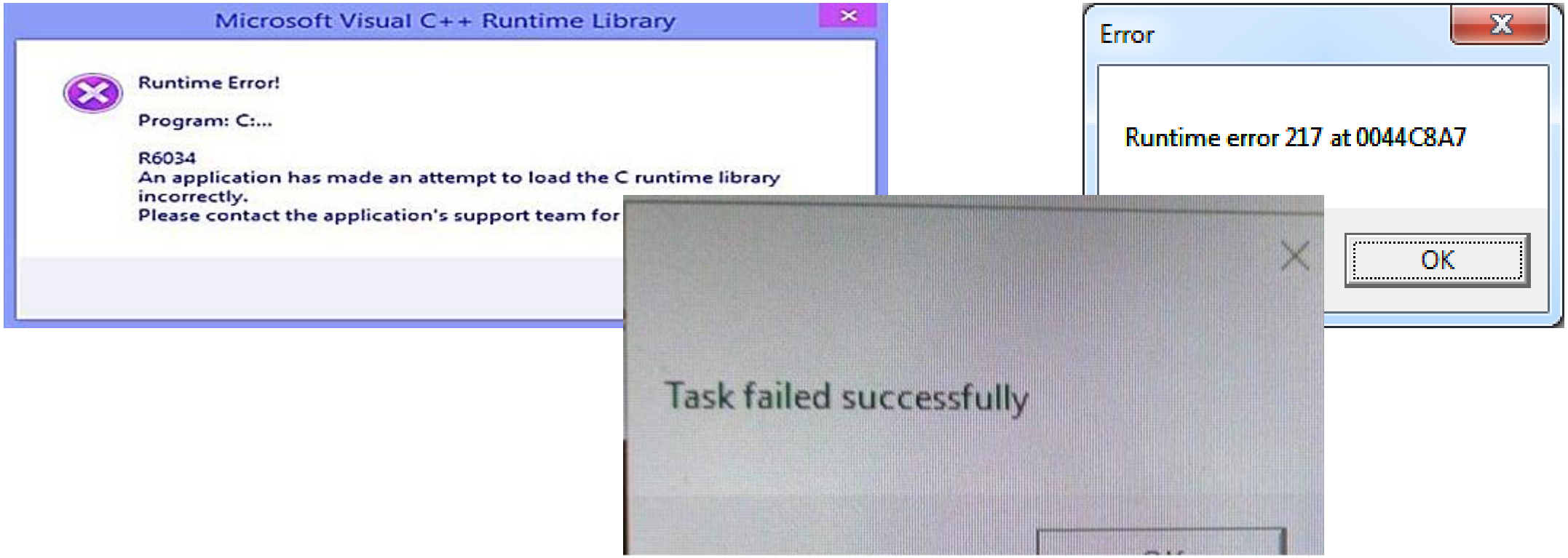
```
>>> while True print 'Hello world'
  File "<stdin>", line 1, in ?
    while True print 'Hello world'
                   ^
SyntaxError: invalid syntax
```

место первой найденной ошибки

Ошибка синтаксиса – ничего не напоминает?

Ошибка синтаксиса

Статический анализ позволяет избежать таких чудес



C# / Java / … – сильная статическая типизация и т.д.

https://habr.com/ru/post/161205/

Python - свобода самовыражения, удобство, читаемость

Исключения - exceptions

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

Исключения - exceptions

Особая ситуация при работе программы, функции, с которой нужно что-то делать

Например:

Исключения - exceptions

Особая ситуация при работе программы, функции, с которой нужно что-то делать

Например:

- целочисленное деление на ноль

Исключения - exceptions

Особая ситуация при работе программы, функции, с которой нужно что-то делать

Например:

- целочисленное деление на ноль
- считывание данных без доступа к ним
- исчерпание доступной памяти
- аварийное отключение электропитания системы
- …

Исключения - exceptions

Исключения – универсальный интерфейс для обработки “неправильных” ситуаций

Как было раньше? Коды возврата

у функции find

видны С-шные корни

```
>>> s = "abc"
>>> s.find("b")
1
>>> s.index("b")
1
>>> s.find("m")
-1
>>> s.index("m")
Traceback (most recent call last):
  File "<pyshell#135>", line 1, in
    s.index("m")
ValueError: substring not found
```

Exception vs error

error

то, что нельзя исправить
можно только сообщить:

- записать в лог
- отправить email
- разработчику извинится перед пользователем.

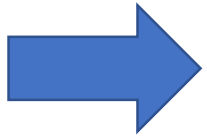
exception

особые ситуации,
которые нужно
как-то **обработать**

Exception vs error

exception

умеют обрабатывать поток,
как, например *if-then-else*

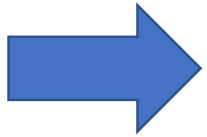


особые ситуации,
которые нужно
как-то **обработать**

пробрасывание по каскаду

```
1  def a(x):
2      try:
3          b(x)
4      except IndexError:
5          print("Well, there's an index error")
6
7  def b(x):
8      c(x)
9
10 def c(x):
11     x[1]
12
13 l = []
14 a(l)
```

```
Well, there's an index error
[Finished in 0.1s]
```



exception

особые ситуации,
которые нужно
как-то **обработать**

Exception

Исключение – особый класс, наследуемый от класса Exception

При выполнении каждой строки интерпретатор проверяет, не вызвалось ли исключение, и затем “пробрасывает его” следующему по стеку вызовов методу.

Он может обработать исключение, либо также “пробросить” его ещё выше

исключение обработано a(x)

```
1  def a(x):
2      try:
3          b(x)
4      except IndexError:
5          print("Well, there's an index error")
6
7  def b(x):
8      c(x)
9
10 def c(x):
11     x[1]
12
13 l = []
14 a(l)
```

```
Well, there's an index error
[Finished in 0.1s]
```

исключение обработано обработчиком Python

```
1  def a(x):
2      b(x)
```

```
File "C:\Users\iom96\Desktop\1.py", line 2, in a
```

```
3
4
5  def b(x):
6      c(x)
```

```
File "C:\Users\iom96\Desktop\1.py", line 6, in b
```

```
7
8  def c(x):
9      x[1]
```

```
File "C:\Users\iom96\Desktop\1.py", line 9, in c
```

```
10
11 l = []
12 a(l)
```

```
File "C:\Users\iom96\Desktop\1.py", line 12, in <module>
```

```
Traceback (most recent call last):
  File "C:\Users\iom96\Desktop\1.py", line 12, in <module>
    a(l)
  File "C:\Users\iom96\Desktop\1.py", line 2, in a
    b(x)
  File "C:\Users\iom96\Desktop\1.py", line 6, in b
    c(x)
  File "C:\Users\iom96\Desktop\1.py", line 9, in c
    x[1]
IndexError: list index out of range
```

Холивар

будьте аккуратны, обрабатывая исключения самостоятельно или ускоряя код с помощью пробрасывания

неявный переход на много вызовов вверх программисту очень тяжело читать и затем обрабатывать в коде

Обработка исключений

```
try:
        # наш обычный код, работающий до наступления исключения
except Exception:
        # срабатывает при наступлении исключения Exception
else:
        # срабатывает после try, если исключение не наступило
finally:
        # срабатывает в конце в любом случае
```

Обработка исключений

```
f = open('1.txt')
ints = []
try:
        for line in f:
                ints.append(int(line))
except ValueError:
        print('Это не число. Выходим.')
except Exception:
        print('Это что ещё такое?')
else:
        print('Всё хорошо.')
finally:
        f.close()
        print('Я закрыл файл.')
# try, группа except, затем else, и только потом finally.
```

Обработка исключений

ValueError? Exception?

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 (обычно сочетанием клавиш Ctrl+C).
- **GeneratorExit** - порождается при вызове метода close объекта generator.
- **Exception** - а вот тут уже заканчиваются полностью системные исключения (которые лучше не трогать) и начинаются обыкновенные, с которыми можно работать.
  - **StopIteration** - порождается встроенной функцией next, если в итераторе больше нет элементов.
  - **ArithmeticError** - арифметическая ошибка.
    - **FloatingPointError** - порождается при неудачном выполнении операции с плавающей запятой. На практике встречается нечасто.
    - **OverflowError** - возникает, когда результат арифметической операции слишком велик для представления. Не появляется при обычной работе с целыми числами (так как python поддерживает длинные числа), но может возникать в некоторых других случаях.
    - **ZeroDivisionError** - деление на ноль.
  - **AssertionError** - выражение в функции assert ложно.
  - **AttributeError** - объект не имеет данного атрибута (значения или метода).

десятки их!

см. https://pythonworld.ru/tipy-dannyx-v-python/isklyucheniya-v-python-konstrukciya-try-except-dlya-obrabotki-isklyuchenij.html

Холивар №2

пишите адекватные обработчики исключительной ситуации

не пишите try-except везде “просто потому что”

Холивар №2

четыре полуторатонных спутника научной программы Cluster

10 лет и 7 миллиардов долларов, потраченных на разработку

37 секунд полета… бабах

Вызов исключений

в любом месте написать:

raise <имя исключения> <аргументы исключения>

например:

raise Exception('spam', 'eggs')

Вызов исключений

```
>>> try:
...     raise Exception('spam', 'eggs')
... except Exception as inst:
...     print type(inst)     # the exception instance
...     print inst.args      # arguments stored in .args
...     print inst           # __str__ allows args to printed directly
...     x, y = inst          # __getitem__ allows args to be unpacked directly
...     print 'x =', x
...     print 'y =', y
...
<type 'instance'>
('spam', 'eggs')
('spam', 'eggs')
x = spam
y = eggs
```

Вызов исключений

вызвать исключение можно и во время его обработки

это хороший вариант: мы и что-то своё сообщаем, и исключение пробрасываем

```
>>> try:
...     raise NameError, 'HiThere'
... except NameError:
...     print 'An exception flew by!'
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
NameError: HiThere
```

Не используйте “bar ecxept”

Иначе в любом созданном цикле вы не сможете прервать исполнение программы

```python
import time

while True:
    try:
        print("Wheeeeee! You can't stop me")
        time.sleep(0.1)
    except:
        print("Oww... Whatever imma keep running")
```

```
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
^COww... Whatever imma keep running
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
^COww... Whatever imma keep running
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
^COww... Whatever imma keep running
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
^COww... Whatever imma keep running
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
^COww... Whatever imma keep running
Wheeeeee! You can't stop me
```

Используйте “except Exception”

Keyboard Interrupt наследуется не от Exception, а от BaseException

```python
while True:
    try:
        print("Wheeeeee! You can't stop me")
        time.sleep(0.1)
    except Exception:
        print("Oww... Whatever imma keep running")
```

```
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
Wheeeeee! You can't stop me
^CTraceback (most recent call last):
  File "two_bare_except.py", line 13, in <mo
dule>
    time.sleep(0.1)
KeyboardInterrupt
(cv)
~/Work/python_mistakes
```

Создание собственных исключений

исключение – класс, наследуемый от Exception

```
1   class MyException(Exception):
2       def __init__(self, value1, value2):
3           self.value1 = value1
4           self.value2 = value2
5
6   try:
7       raise MyException("Oops!", "That's bad!")
8   except MyException as e:
9       # для работы с исключением можно назвать его более коротко
10      print(f"My Exception occured, {e.args}")
```

```
My Exception occured, ('Oops!', "That's bad!")
[Finished in 0.1s]
```

Создание собственных исключений

исключение – класс, наследуемый от Exception

```
1   class MyException(Exception):
2       def __init__(self, value1, value2):
3           self.value1 = value1
4           self.value2 = value2
5
6       # можно в классе описать, что делать
7       # если Python будет пытаться неявно
8       # преобразовать исключение в строку
9       def __str__(self):
10          return repr(self.value1 + " " + self.value2)
11  try:
12      raise MyException("Oops!", "That's bad!")
13  except MyException as e:
14      # для работы с исключением можно назвать его более коротко
15      print(e)
```

```
"Oops! That's bad!"
[Finished in 0.1s]
```

задание

1. *Внутри функции avrg сделать проверку на неотрицательность произведения a и b иначе вывести ValueError(“…”)*

2. *При печати результата функции avg сделать проверку на вывести ValueError, предложив ввести другие числа Затем сделать проверку на произвольный Exception, выведя его на экран*