БИБЛИОТЕКИ ЯЗЫКА PYTHON ДЛЯ КОМПЬЮТЕРНЫХ ВЫЧИСЛЕНИЙ И МОДЕЛИРОВАНИЯ

Словари, множества, встроенные библиотеки, исключения и файлы в языке Python

Лобанов Алексей Владимирович
Главный специалист отдела разработки и внедрения АИС,
Ассистент кафедры информационных компьютерных технологий РХТУ им. Д.И. Менделеева

ТЕМЫ

- Словари и множества
- Встроенные функции
- Особенности импорта модулей в Python
- Модуль **datetime**
- Модуль **itertools**
- Библиотека **math**
- Модуль **random**
- Работа с файлами
- Исключения и способы их обработки

ПОЛЕЗНЫЕ РЕСУРСЫ

- Коллекции официальная документация
- Словари
- Встроенные функции документация
- Модуль datetime
- Модуль itertools
- Библиотека math
- Модуль random
- Исключения документация
- Файлы
- Формат JSON

СЛОВАРИ

Словари в Python - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами. Объявить словарь можно с помощью литерала {} или функции **dict.**

- Словарь похож на список. Но адресация элементов в нём обеспечивается идентификаторами-ключами.
- Ключ может являться булевой переменной, целым числом, числом с плавающей точкой, кортежем, строкой и другими объектами
- Словарь – изменяемый элемент. Можно добавлять, удалять и изменять его элементы.
- В Python допускается наличие запятой после последнего элемента списка, кортежа или словаря.
- В других языках программирования словари могут называться ключевыми массивами, ассоциативными массивами, хешами или хеш-таблицей

СЛОВАРИ

Словарь обозначается фигурными скобками {}

```
d = {}          # пустой словарь
d ={"Sub":"Hg", "Property":"Metal"}
```

"Sub", "Property" – ключи

```
d = {"key1": 1, "key2": 2} # {'key1': 1, 'key2': 2}
```

Использование метода ***setdefault***

```
d.setdefault("key4", 5) # {'key1': 1, 'key2': 2, 'key4': 5}
d.setdefault("key1", 5) # {'key1': 1, 'key2': 2}
e = d.setdefault("key4", 5) # 5
e = d.setdefault("key1", 5) # 1
```

Создание словаря по ключам

```
d = dict(sh="d", lng="di") # {'sh': 'd', 'lng': 'di'}
```

С помощью метода ***fromkeys***

```
mydict = dict.fromkeys(["a", "b"], 1) # {'a': 1, 'b': 1}
d = dict.fromkeys (["a", "b"] # {'a': None, 'b': None}
```

СЛОВАРИ

Создание словаря из списка списков

```
rawlist = [["a", "b"], ["c", "d"], ["e", "f"]]
d = dict(rawlist)  # {'a': 'b', 'c': 'd', 'e': 'f'}
```

из списка кортежей:

```
rawlist = [("a", "b"), ("c", "d"), ("e", "f")]
d = dict(rawlist) # {'a': 'b', 'c': 'd', 'e': 'f'}
```

из кортежа списков:

```
rawtuple = (["a", "b"], ["c", "d"], ["e", "f"])
d = dict(rawtuple) # {'a': 'b', 'c': 'd', 'e': 'f'}
```

Список строк

```
s = ["ab", "cd", "ef"]
d = dict(s)          # {'a': 'b', 'c': 'd', 'e': 'f'}
```

Кортеж строк

```
s = ("ab", "cd", "ef")
d = dict(s)          # {'a': 'b', 'c': 'd', 'e': 'f'}
```

СЛОВАРИ (МЕТОДЫ)

dict.clear() - очищает словарь.

dict.copy() - возвращает копию словаря.

classmethod **dict.fromkeys**(seq[, value]) - создает словарь с ключами из seq и значением value (по умолчанию None).

dict.get(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).

dict.items() - возвращает пары (ключ, значение).

dict.keys() - возвращает ключи в словаре.

dict.pop(key[, default]) - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).

dict.popitem() - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.

dict.setdefault(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).

dict.update([other]) - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).

dict.values() - возвращает значения в словаре.

СЛОВАРИ (РАСПАКОВКА)

Распаковка может быть применена и для словарей. С помощью распаковки можно легко объединить несколько списков. В случае наличия одинаковых элементов, присваиваются значения наиболее позднего вхождения. Иногда с использованием распаковки удобно передавать параметры в функцию (смотри в следующей серии…).

Объединение словарей распаковкой

```
1 a = {"a": 1, "b": 2}
2 b = {"b": 3, "c": 4}
3 c = {**a, **b}
4 print(c)
```

СЛОВАРИ (ПЕРЕЧИСЛЕНИЕ)

По умолчанию перечисление словарей осуществляется только по ключам. Однако, с помощью автоматической распаковки на каждой итерации цикла, можно осуществлять одновременное перечисления по ключам и значениям словаря

Без распаковки

```
1 a = {"a": 1, 2: "b", 3.5: [1, 2]}
2 # Только по ключам
3 for el in a:
4     print(el)
5 # С обращением по ключам
6 for el in a:
7     print(el, a[el])
```

С распаковкой

```
1 a = {"a": 1, 2: "b", 3.5: [1, 2]}
2 for k, v in a.items():
3     print(k, v)
```

МНОЖЕСТВА (SET)

Множество похоже на словарь, но имеет только ключи, а значения опущены.

Ключи должны быть уникальными.

Порядок ключей не имеет значения.

- Создание пустого множества

empty_set = set()

Примеры создания множеств:

set("text") # {'t', 'e', 'x'}

- из списка

d = set(["Раз", "Два", "Два", "Три"])

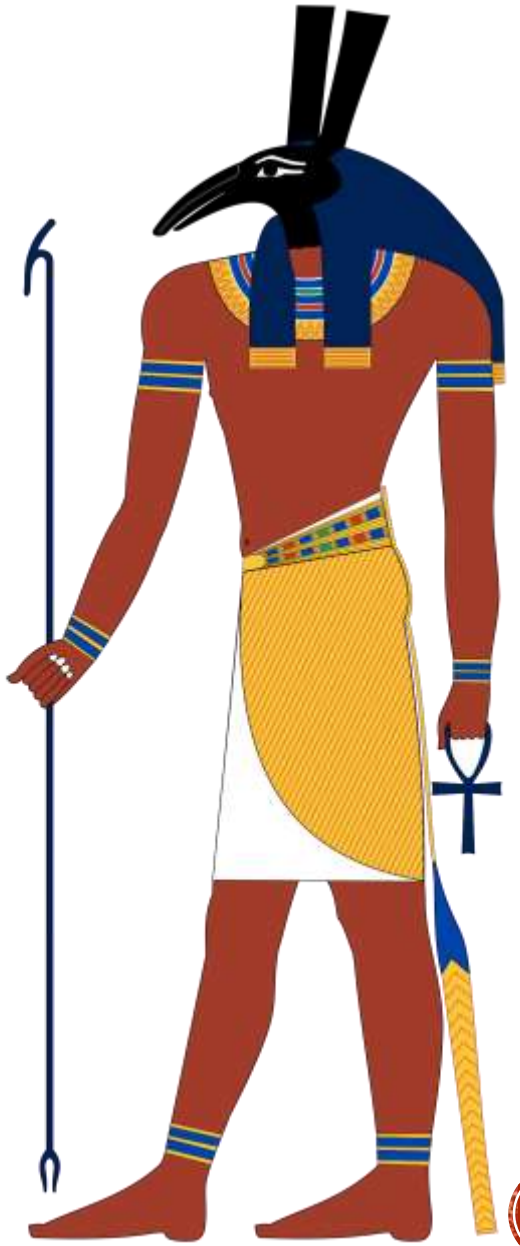
 # {'Два', 'Раз', 'Три'} (порядок может меняться)

- из кортежа

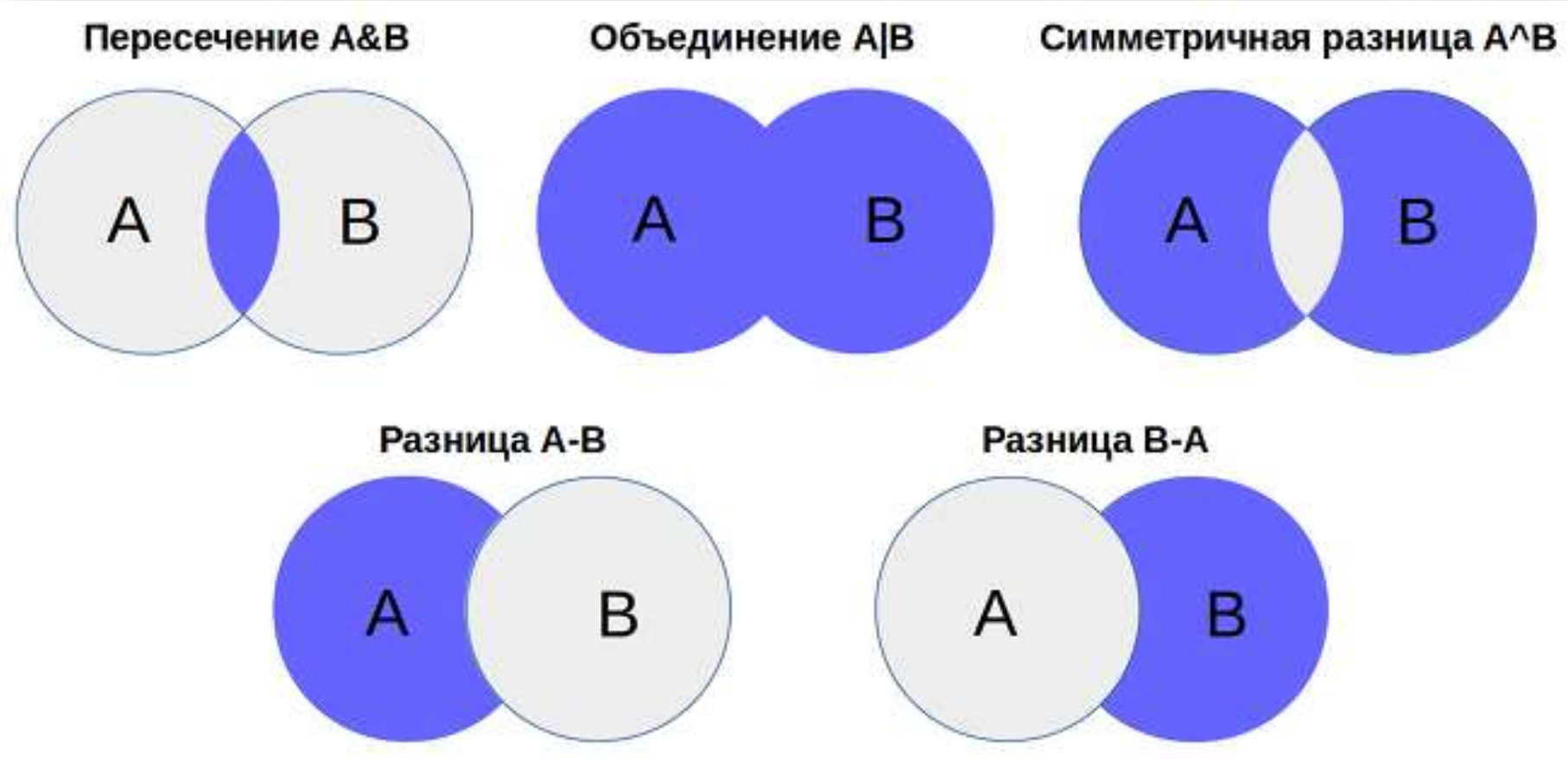
d= set(("Раз", "Два", "Два", "Три"))

 # {'Два', 'Раз', 'Три'} (порядок может меняться)

d= set({"a":1, "b": 2, "c": 3}) # {'c', 'b', 'a'}



МНОЖЕСТВА (SET)



МНОЖЕСТВА (SET)

```
d = set(["a", "b", "c"])
e = set(["c", "d", "e"])
```

- пересечение множеств (**&**, **intersection**):

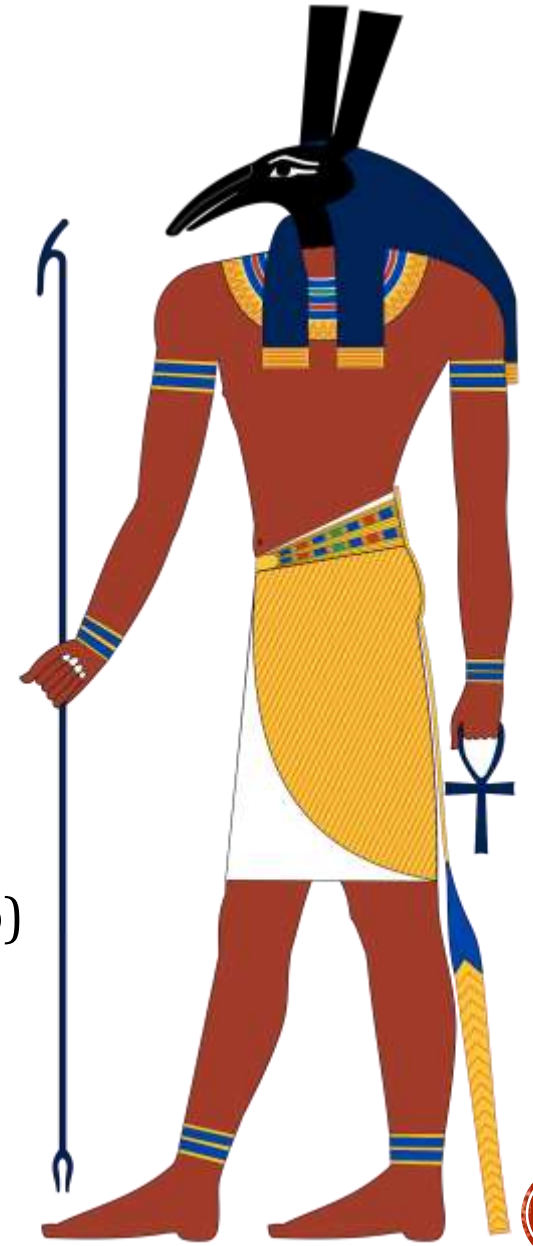
```
f =  d & e                          # {'c'}
g = d.intersection(e)     # {'c'}
```

- объединение множеств (**|** , **union**)

```
f =  d | e                          # {'b', 'a', 'd', 'c', 'e'}
g = d.union(e)              # {'b', 'a', 'd', 'c', 'e'}
```

- Разность множеств (члены только первого множества, но не второго) **(–, difference)**

```
f =  d – e                          # {'a', 'b'}
g = d.difference(e)         # {'a', 'b'}
```



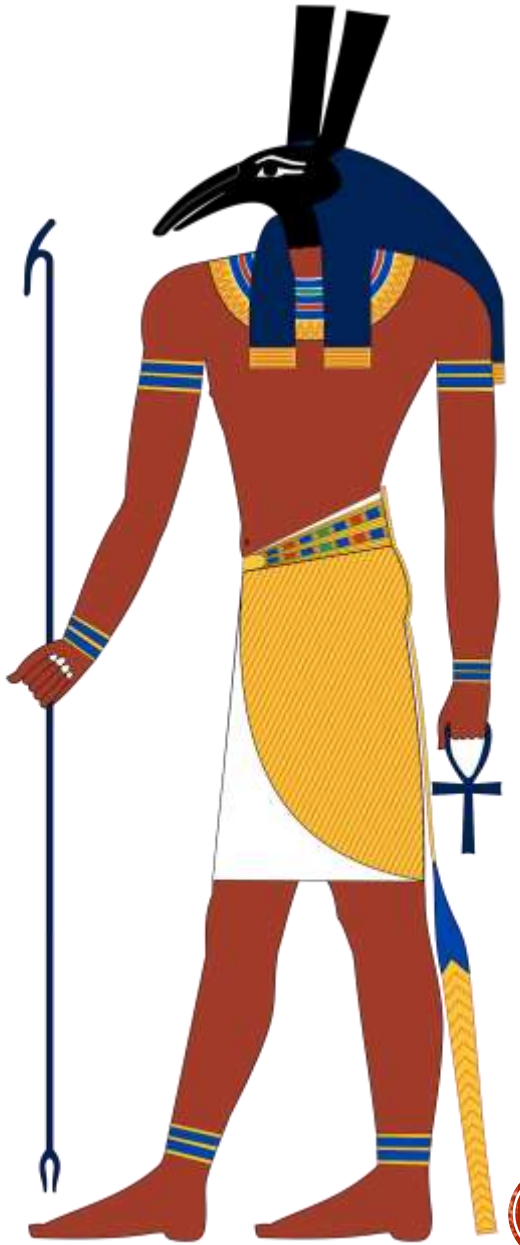
МНОЖЕСТВА (SET)

```
d = set(["a", "b", "c"])
e = set(["c", "d", "e"])
```

- исключающее ИЛИ (элементы или первого, или второго множества, но не общие)

(^ , **symmetric_difference()**)

```
f =  d ^ e                                  # {'b', 'd', 'e', 'a'}
g = d.symmetric_difference(e)               # {'b', 'd', 'e', 'a'}
```



МНОЖЕСТВА (МЕТОДЫ)

a.isdisjoint(b)	Истина, если ***a*** и ***b*** не имеют общих элементов.
a == b	Все элементы ***a*** принадлежат ***b***, все элементы ***b*** принадлежат ***a***.
a.issubset(b) ***a <= b***	Все элементы ***a*** принадлежат ***b***.
a.issuperset(b) ***a >= b***	Все элементы ***b*** принадлежат ***a***.
a.union(b, ...) ***a | b | ...***	Объединение нескольких множеств.
a.intersection(b, ...) ***a & b & ...***	Пересечение.
a.difference(b, ...) ***a – b – …***	Множество из всех элементов ***a***, не принадлежащие ни одному из ***b***.
a.symmetric_difference(b) ***a^ b***	Множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.

МНОЖЕСТВА (МЕТОДЫ)

a.copy()	Копия множества.
a.update(b, ...) ***a |= b | ...***	Объединение.

a.intersection_update(b, ...); ***a &= b & ...***	Пересечение.
set.difference_update(other, ...); set – = other | …	Вычитание.
a.symmetric_difference_update(b); ***set ^= other***	Множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
a.add(element)	Добавляет элемент в множество.
a.remove(element)	Удаляет элемент из множества. Возникает исключение **KeyError**, если такого элемента не существует.

МНОЖЕСТВА (МЕТОДЫ)

a.discard(element)	Удаляет элемент, если он находится в множестве.
a.pop()	Удаляет первый элемент из множества. Так как множества не упорядочены, нельзя точно сказать, какой элемент будет первым.
a.clear()	Очистка множества.

МНОЖЕСТВА (SET И FROZENSET)

Действия над множествами

Добавление элемента во множество:

```
d = set(["a1", "b2", "c3"])
d.add("d4")        # {'a1', 'b2', 'c3', 'd4'}
```

Проверка наличия элемента:

```
e = "a" in d       # False
e = "b2" in d          # True
```

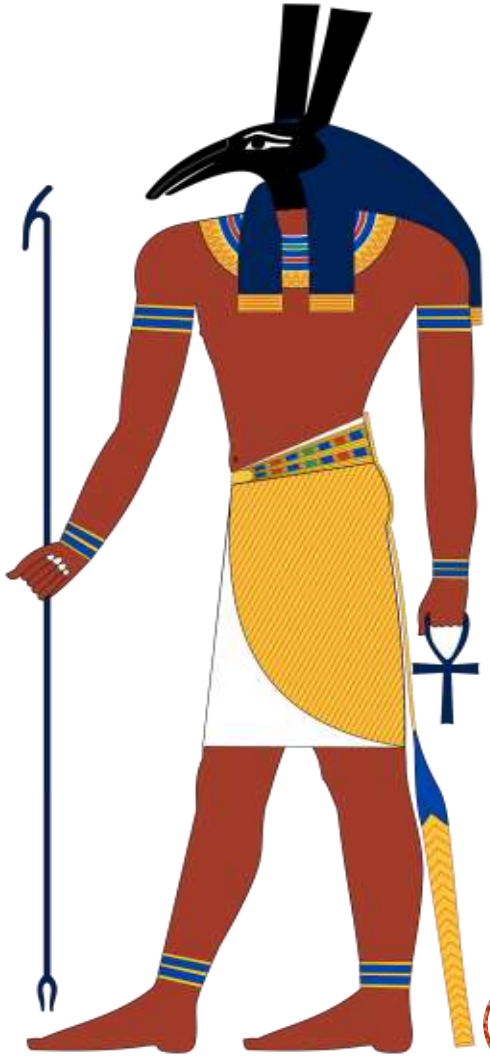
Удаление элемента из множества:

```
d.remove("a1")  # {'b2', 'c3', 'd4'}
d.discard("b2")  # {'c3', 'd4'}
```

Метод ***discard()*** не выдает ошибку, если элемента нет во множестве в отличие от метода ***remove()***.

set – изменяемый тип данных, а ***frozenset*** – нет.
Примерно схожая ситуация со списками и кортежами.

```
b = frozenset("текст")
b.add(1)           # ОШИБКА
```



ВСТРОЕННЫЕ ФУНКЦИИ

Встроенные функции - это функции, которые доступны для пользователя с самого начала без импортирования дополнительных библиотек. Всего существует 69 встроенных функций, однако не все имеют одинаковую значимость для решения вычислительных задач. Далее рассматриваются 24 наиболее полезные встроенные функции.

ВСТРОЕННЫЕ ФУНКЦИИ

input

```
res1 = input()
print(res1)
res2 = input("Enter data: ")
print(res2)
```

str

```
res1 = str(1.5)
print(res1)

res2 = str([1, 2])
print(res2)

res3 = str(abs)
print(res3)
```

list

```
res1 = list()
pritn(res1)
res2 = list((1, 2, 3))
print(res2)
res3 = list("abc")
print(res3)
res4 = list({"a": 1, "b": 2})
print(res4)
```

tuple

```
res1 = tuple()
pritn(res1)
res2 = tuple([1, 2, 3])
print(res2)
res3 = tuple("abc")
print(res3)
res4 = tuple({"a": 1, "b": 2})
print(res4)
```

len

```
a = [1, 2, 3, 4]
b = {"a": 1, "b": 2}
print(len(a))
print(len(b))
```

int

```
res1 = int()
print(res1)
res2 = int(2.5)
print(res2)
res3 = int("125")
print(res3)
# Преобразование из других
# систем счисления
res4 = int("FF", base=16)
print(res4)
```

print

```
print(1)
print(1, 2, 3)
print(1, 2, 3, sep=", ")
print(1, 2, 3, sep=", ", end=".\n")
```

float

```
res1 = float()
print(res1)
res2 = float(1)
print(res2)
res2 = float("3.7")
print(res2)
```

ВСТРОЕННЫЕ ФУНКЦИИ

Функция **abs** возвращает модуль числа или магнитуду комплексного числа. Функция **all** возвращает True если все элементы коллекции приводятся к True, иначе False.

```
res1 = abs(-4)
print(res1)

res2 = abs(1 - 5j)
print(res2)
```

```
4
5.099019513592784
```

```
res1 = all([True, True, False])
print(res1)
res2 = all([True, 1, "abc"])
print(res2)
res3 = all([-1, 2, 0, 3, 1])
print(res3)
```



```
False
True
False
```

ВСТРОЕННЫЕ ФУНКЦИИ

Функция **any** возвращает True если хотя бы один элемент коллекции приводится к True, иначе False. Функция **all** используется для преобразования различных типов данных в комплексное число. В случае преобразовании строки в комплексное число, строка не должна содержать пробелов между числами и плюсом.

```
res1 = all(["", 0, False])
print(res1)
res2 = all([True, 1, "abc"])
print(res2)
res3 = all([-1, 2, 0, 3, 1])
print(res3)
```

```
False
True
True
```

```
res1 = complex()
print(res1)
res2 = complex(1)
print(res2)
res2 = complex(1, 2)
print(res2)
res2 = complex("1+2j")
print(res2)
```

```
0j
(1+0j)
(1+2j)
(1+2j)
```

ВСТРОЕННЫЕ ФУНКЦИИ

Функция **divmod** возвращает результат целочисленного деления и остаток от деления. Функция **enumerate** используется для перечисления по коллекции с отслеживанием текущего индекса элемента.

```
res = divmod(7, 2)
pritn(res)
```

→

```
(3, 1)
```

```
a = [1, 2.5, "a", True]
print("From 0:")
for i, v in enumerate(a):
    print(f"{i}: {v}")
print("From 5:")
for i, v in enumerate(a, start=5):
    print(f"{i}: {v}")
```

→

```
From 0:
0) 1
1) 2.5
2) 'a'
3) True
From 5:
5) 1
6) 2.5
7) 'a'
8) True
```

ВСТРОЕННЫЕ ФУНКЦИИ

Функция **help** выводит справку по какому либо объекту. Функция **map** Применяет указанную функцию к элементам коллекции (или нескольких коллекций). Функция map возвращает объект map с которым можно работать непосредственно в цикле, либо преобразовать его к списку или кортежу.

```
help(2)
```

```
Help on int object:

class int(object)
 |  int([x]) -> integer
 |  int(x, base=10) -> integer
...
```

```
a = [1, 2, 3]
b = [4, 5 ,6]
m = map(lambda: x: x**2, a)
res1 = list(m)
print(res1)

m = map(lambda x1, x2: x1 + x2, a, b)
res2 = list(m)
print(res2)
```

```
[1, 4, 9]
[5, 7, 9]
```

ВСТРОЕННЫЕ ФУНКЦИИ

Функции **min** и **max** используются для нахождения минимума или максимума коллекции, соответственно. Функция **round** используется для округления чисел с плавающей точкой с учетом правил математики.

```
a = [1, 2, -3, 0, -5, 6]

res1 = min(a)
print(res1)

res2 = min(a, key=abs)
print(res2)
```

```
-5
0
```

```
res1 = round(4.576345)
pritn(res1)

res2 = round(4.576345, 2)
print(res2)
```

```
5
4.58
```

ВСТРОЕННЫЕ ФУНКЦИИ

Функции **sorted** используется для сортировки элементов коллекции. Функция **sum** возвращает сумму элементов коллекции. Второй аргумент отвечает за число к которому будут прибавляться элементы коллекции (по умолчанию 0).

```
a = [1, 2, -3, 0, -5, 6]

res1 = sorted(a)
print(res1)

res2 = sorted(a, key=abs)
print(res2)

res3 = sorted(a,
              key=abs,
              reverse=True)
print(res3)
```



```
[-5, -3, 0, 1, 2, 6]
[0, 1, 2, -3, -5, 6]
[6, -5, -3, 2, 1, 0]
```

```
a = [1, 2, 3, 4]

res1 = sum(a)
print(res1)

res2 = sum(a, 10)
print(res2)
```



```
10
20
```

ВСТРОЕННЫЕ ФУНКЦИИ

Функции **type** возвращает тип указанного объекта. Функция **zip** позволяет реализовывать одновременное перечисление по нескольким коллекциям. перечисление осуществляется до конца самой короткой коллекции.

```
res1 = type(2)
print(res1)

res2 = type(0.5)
print(res2)

res3 = type(abs)
print(res3)

res4 = type(5) == int
print(res4)
```



```
<class 'int'>
<class 'float'>
<class 'builtin_function_or_method'>
True
```

```
a = [1, 2, 3, 4]
b = [5, 6, 7, 8, 9, 10]

for i, j, k in zip(a, b, a):
    print(f"{i} - {j} - {k}")
```



```
1 - 5 - 1
2 - 6 - 2
3 - 7 - 3
4 - 8 - 4
```

ИМПОРТ МОДУЛЕЙ

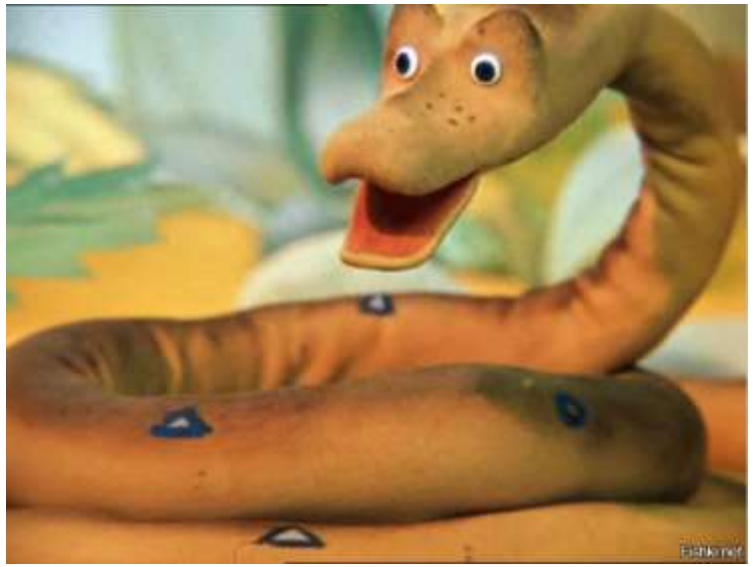
Подключить модуль можно с помощью инструкции import. осле ключевого слова import указывается название модуля. Одной инструкцией можно подключить несколько модулей, хотя этого не рекомендуется делать, так как это снижает читаемость кода. После импортирования модуля его название становится переменной, через которую можно получить доступ к атрибутам модуля. Если название модуля слишком длинное, или оно вам не нравится по каким-то другим причинам, то для него можно создать псевдоним, с помощью ключевого слова as. Подключить определенные атрибуты модуля можно с помощью инструкции from

```
#импортируем модуль, datetime можем использовать как переменную
import datetime

#сокращаем имя через псевдонимы
import itertools as it

#импортируем всё (не рекомендуется)
from json import *

#импортируем отдельные атрибуты и используем псевдонимы
from math import e, ceil as c
```



МОДУЛЬ DATETIME

Модуль datetime предоставляет классы для обработки времени и даты разными способами. Поддерживается и стандартный способ представления времени, однако больший упор сделан на простоту манипулирования датой, временем и их частями.

МОДУЛЬ DATETIME (ОСНОВНЫЕ ТИПЫ)

- **datetime.date(year, month, day)** - стандартная дата. Атрибуты: **year, month, day.** Неизменяемый объект
- **datetime.time(hour=0, minute=0, second=0, microsecond=0, tzinfo=None)** - стандартное время, не зависит от даты. Атрибуты: **hour, minute, second, microsecond, tzinfo.**
- **datetime.timedelta** - разница между двумя моментами времени, с точностью до микросекунд.
- **datetime.datetime**(year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None) - комбинация даты и времени

МОДУЛЬ DATETIME (ОСНОВНЫЕ ФУНКЦИИ)

- **datetime.today()** – текущая дата и время.
- **datetime.combine**(date, time) – datetime из комбинации объектов date и time.
- **datetime.strptime**(date_string, format) – преобразует строку в datetime.
- **datetime.strftime**(format) – преобразует datetime в строку.
- **datetime.weekday**() – день недели в виде числа, понедельник - 0, воскресенье - 6.

МОДУЛЬ ITERTOOLS

Данный модуль является сборником полезных итераторов, повышающих эффективность работы с циклами и генераторами последовательностей объектов. Это достигается за счет лучшего управления памятью в программе, быстрого выполнения подключаемых функций, а также сокращения и упрощения кода. Готовые методы, реализованные в данной библиотеке, принимают различные параметры для управления генератором последовательности, чтобы вернуть вызывающей подпрограмме необходимый набор объектов.

МОДУЛЬ ITERTOOLS (ФУНКЦИИ)

itertools.count(start=0, step=1) - бесконечная арифметическая прогрессия с первым членом start и шагом step.

itertools.cycle(iterable) - возвращает по одному значению из последовательности, повторенной бесконечное число раз.

itertools.repeat(elem, n=Inf) - повторяет elem n раз.

itertools.accumulate(iterable) - аккумулирует суммы.

itertools.chain(*iterables) - возвращает по одному элементу из первого итератора, потом из второго, до тех пор, пока итераторы не кончатся.

itertools.combinations(iterable, [r]) - комбинации длиной r из iterable без повторяющихся элементов.

itertools.combinations_with_replacement(iterable, r) - комбинации длиной r из iterable с повторяющимися элементами.

МОДУЛЬ ITERTOOLS (ФУНКЦИИ)

itertools.filterfalse(func, iterable) - все элементы, для которых func возвращает ложь.

itertools.groupby(iterable, key=None) - группирует элементы по значению. Значение получается применением функции key к элементу (если аргумент key не указан, то значением является сам элемент).

itertools.permutations(iterable, r=None) - перестановки длиной r из iterable.

itertools.product(*iterables, repeat=1) - аналог вложенных циклов.

itertools.starmap(function, iterable) - применяет функцию к каждому элементу последовательности (каждый элемент распаковывается).

itertools.takewhile(func, iterable) - элементы до тех пор, пока func возвращает истину.

itertools.tee(iterable, n=2) - кортеж из n итераторов.

itertools.zip_longest(*iterables, fillvalue=None) - как встроенная функция zip, но берет самый длинный итератор, а более короткие дополняет fillvalue.

БИБЛИОТЕКА MATH

Python библиотека math содержит наиболее применяемые математические функции и константы. Все вычисления происходят на множестве вещественных чисел.

math.ceil(X) – округление до ближайшего большего числа.

math.copysign(X, Y) - возвращает число, имеющее модуль такой же, как и у числа X, а знак - как у числа Y.

math.fabs(X) - модуль X.

math.factorial(X) - факториал числа X.

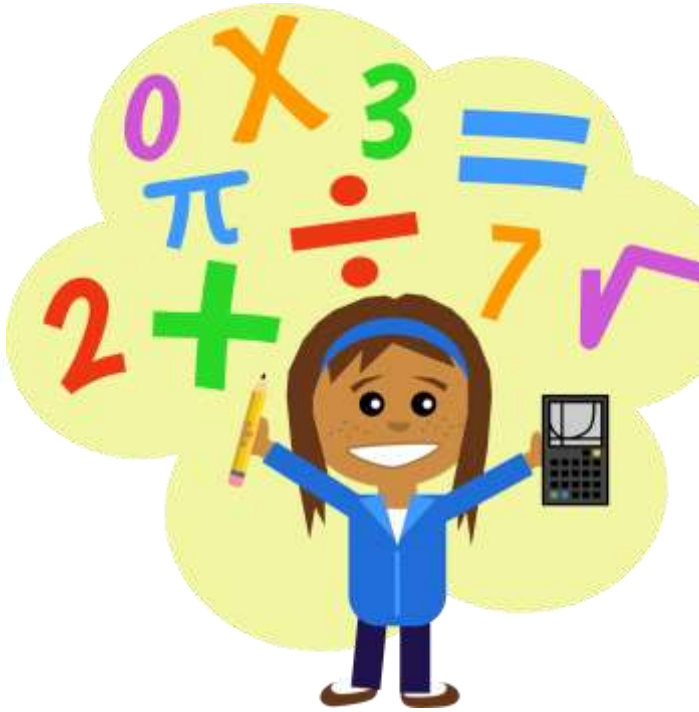
math.floor(X) - округление вниз.

math.fmod(X, Y) - остаток от деления X на Y.

math.frexp(X) - возвращает мантиссу и экспоненту числа.

math.ldexp(X, I) - $X * 2^i$. Функция, обратная функции math.frexp().

БИБЛИОТЕКА MATH



math.fsum(последовательность) - сумма всех членов последовательности. Эквивалент встроенной функции sum(), но math.fsum() более точна для чисел с плавающей точкой.

math.isfinite(X) - является ли X числом.

math.isinf(X) - является ли X бесконечностью.

math.isnan(X) - является ли X NaN (Not a Number - не число).

math.modf(X) - возвращает дробную и целую часть числа X. Оба числа имеют тот же знак, что и X.

math.trunc(X) - усекает значение X до целого.

math.exp(X) - e^X.

math.expm1(X) - $e^X - 1$. При $X \to 0$ точнее, чем math.exp(X)-1.

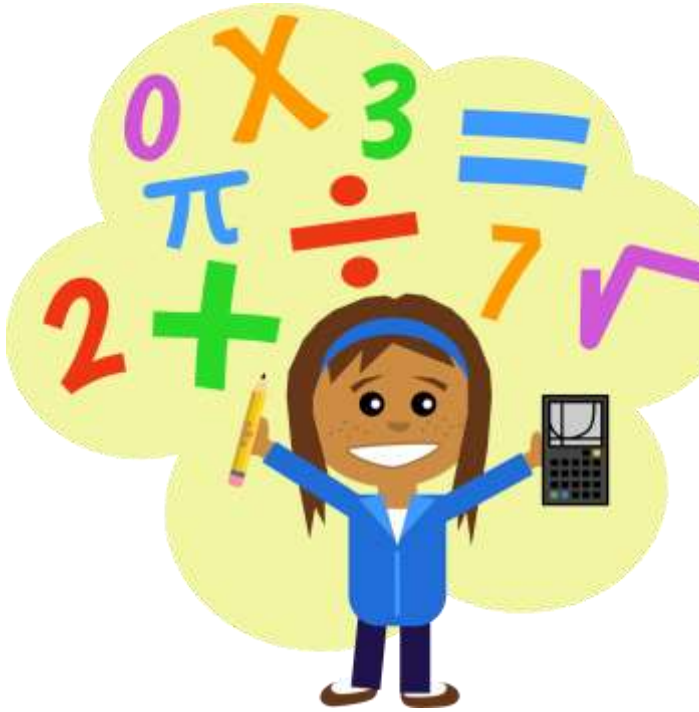
math.log(X, [base]) - логарифм X по основанию base. Если base не указан, вычисляется натуральный логарифм.

math.log1p(X) - натуральный логарифм (1 + X). При $X \to 0$ точнее, чем math.log(1+X).

math.log10(X) - логарифм X по основанию 10.

math.log2(X) - логарифм X по основанию 2.

БИБЛИОТЕКА MATH



math.pow(X, Y) - X^Y.

math.sqrt(X) - квадратный корень из X.

math.acos(X) - арккосинус X. В радианах.

math.asin(X) - арксинус X. В радианах.

math.atan(X) - арктангенс X. В радианах.

math.atan2(Y, X) - арктангенс Y/X. В радианах. С учетом четверти, в которой находится точка (X, Y).

math.cos(X) - косинус X (X указывается в радианах).

math.sin(X) - синус X (X указывается в радианах).

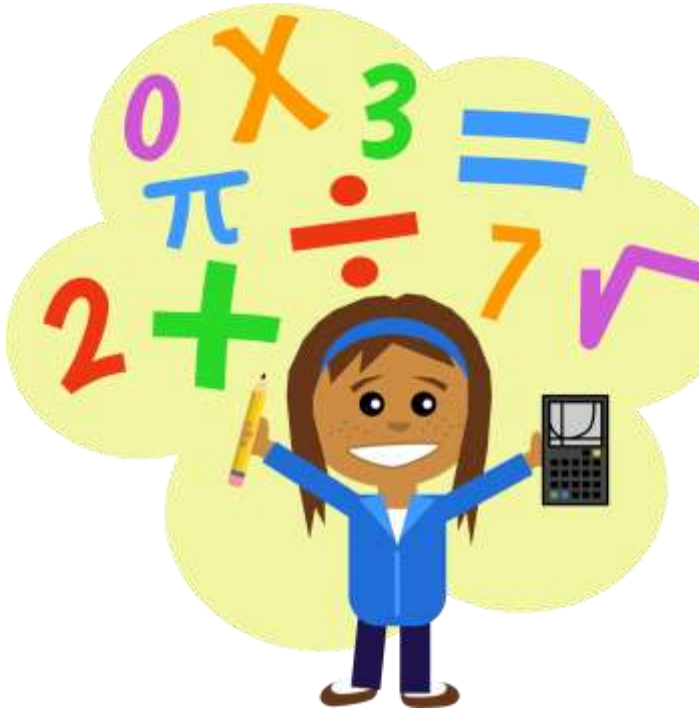
math.tan(X) - тангенс X (X указывается в радианах).

math.hypot(X, Y) - вычисляет гипотенузу треугольника с катетами X и Y (math.sqrt(x * x + y * y)).

math.degrees(X) - конвертирует радианы в градусы.

math.radians(X) - конвертирует градусы в радианы.

БИБЛИОТЕКА MATH



math.cosh(X) - вычисляет гиперболический косинус.

math.sinh(X) - вычисляет гиперболический синус.

math.tanh(X) - вычисляет гиперболический тангенс.

math.acosh(X) - вычисляет обратный гиперболический косинус.

math.asinh(X) - вычисляет обратный гиперболический синус.

math.atanh(X) - вычисляет обратный гиперболический тангенс.

math.erf(X) - функция ошибок.

math.erfc(X) - дополнительная функция ошибок (1 - math.erf(X)).

math.gamma(X) - гамма-функция X.

math.lgamma(X) - натуральный логарифм гамма-функции X.

math.pi - pi = 3,1415926...

math.e - e = 2,718281...

МОДУЛЬ RANDOM

Модуль random предоставляет функции для генерации случайных чисел, букв, случайного выбора элементов последовательности.

random.randrange(start, stop, step) - возвращает случайно выбранное число из последовательности.

random.randint(A, B) - случайное целое число N, A ≤ N ≤ B.

random.choice(sequence) - случайный элемент непустой последовательности.

random.shuffle(sequence, [rand]) - перемешивает последовательность (изменяется сама последовательность). Поэтому функция не работает для неизменяемых объектов.

random.sample(population, k) - список длиной k из последовательности population.

random.random() - случайное число от 0 до 1.

random.uniform(A, B) - случайное число с плавающей точкой, A ≤ N ≤ B (или B ≤ N ≤ A).

МОДУЛЬ RANDOM

random.triangular(low, high, mode) - случайное число с плавающей точкой, low ≤ N ≤ high. Mode - распределение.

random.betavariate(alpha, beta) - бета-распределение. alpha>0, beta>0. Возвращает от 0 до 1.

random.expovariate(lambd) - экспоненциальное распределение. lambd равен 1/среднее желаемое. Lambd должен быть отличным от нуля. Возвращаемые значения от 0 до плюс бесконечности, если lambd положительно, и от минус бесконечности до 0, если lambd отрицательный.

random.gammavariate(alpha, beta) - гамма-распределение. Условия на параметры alpha>0 и beta>0.

random.gauss(значение, стандартное отклонение) - распределение Гаусса.

random.lognormvariate(mu, sigma) - логарифм нормального распределения. Если взять натуральный логарифм этого распределения, то вы получите нормальное распределение со средним mu и стандартным отклонением sigma. mu может иметь любое значение, и sigma должна быть больше нуля.

random.normalvariate(mu, sigma) - нормальное распределение. mu - среднее значение, sigma - стандартное отклонение.

random.vonmisesvariate(mu, kappa) - mu - средний угол, выраженный в радианах от 0 до 2π, и kappa - параметр концентрации, который должен быть больше или равен нулю. Если каппа равна нулю, это распределение сводится к случайному углу в диапазоне от 0 до 2π.

random.paretovariate(alpha) - распределение Парето.

random.weibullvariate(alpha, beta) - распределение Вейбулла.

ФАЙЛЫ

Файлы помогают программисту упростить процедуру ввода и сохранения различной информации, необходимой для работоспособности приложения.

Для открытия файлов используется встроенная функция open, в неё передается путь к файлу и режим открытия.

```
f = open("file.txt", "r")
```

Путь к файл Режим открытия

Режим	Обозначение
'r'	открытие на чтение (является значением по умолчанию).
'w'	открытие на запись, содержимое файла удаляется, если файла не существует, создается новый.
'x'	открытие на запись, если файла не существует, иначе исключение.
'a'	открытие на дозапись, информация добавляется в конец файла.
'b'	открытие в двоичном режиме.
't'	открытие в текстовом режиме (является значением по умолчанию).
'+'	открытие на чтение и запись

ФАЙЛЫ (СОСТАВНЫЕ РЕЖИМЫ)

"rb"	Открывает файл для чтения в двоичном формате. Указатель стоит в начале файла.
"r+"	Открывает файл для чтения и записи. Указатель стоит в начале файла.
"rb+"	Открывает файл для чтения и записи в двоичном формате. Указатель стоит в начале файла.
"wb"	Открывает файл для записи в двоичном формате. Указатель стоит в начале файла. Создает файл с именем ***file***, если такового не существует.
"w+"	Открывает файл для чтения и записи. Указатель стоит в начале файла. Создает файл с именем ***file***, если такового не существует.
"wb+"	Открывает файл для чтения и записи в двоичном формате. Указатель стоит в начале файла. Создает файл с именем ***file***, если такового не существует.
"ab"	Открывает файл для добавления в двоичном формате. Указатель стоит в конце файла. Создает файл с именем ***file***, если такового не существует.
"a+"	Открывает файл для добавления и чтения. Указатель стоит в конце файла. Создает файл с именем ***file***, если такового не существует.
"ab+"	Открывает файл для добавления и чтения в двоичном формате. Указатель стоит в конце файла. Создает файл с именем ***file***, если такового не существует.

ФАЙЛЫ(ЧТЕНИЕ И ЗАПИСЬ)

Чтение из файла осуществляется при помощи функции read, в неё можно передать количество символов, которое нужно считать, если не передавать ничего, считается всё из файла. Для записи в файл применяется функция write. После работы с файлом рекомендуется закрыть функцией close.

Запись данных в файл

```
>>> f = open("file.txt", 'w')
>>> stroka = "qwerty"
>>> f.write(stroka)
>>> f.close()
```

Чтение данных из файла

```
>>> f = open("file.txt", 'r')
>>> print(f.read())
qwerty
>>> f.close()
```

ФАЙЛЫ(ЗАКРЫТИЕ)

Для рационального использования ресурсов, рекомендуется всегда закрывать файлы, предпочтительно и наиболее просто это делается при помощи конструкции with.

Использование **with**

```
with open("file.txt", "r") as f:
    print(f.read())
```

ФАЙЛЫ(ПРИМЕР ЗАПИСИ)

```
import csv
a = [
    ["Аня", "Иванова"],
    ["Таня", "Смирнова"],
    ["Рита", "Кузнецова"],
    ["Стас", "Ежов"],
    ["Оля", "Бялко"],]
with open("out.csv", "wt", encoding="utf-8") as fout: # менеджер контекста
    csvout = csv.writer(fout)
    csvout.writerows(a)
```

В файле:

Аня,Иванова

Таня,Смирнова

Рита,Кузнецова

Стас,Ежов

Оля,Бялко

ФАЙЛЫ(ФОРМАТ JSON)

JSON (JavaScript Object Notation) - простой формат обмена данными, основанный на подмножестве синтаксиса JavaScript. Модуль json позволяет кодировать и декодировать данные в удобном формате. По сути формат JSON – это хранение данных в виде словарей. Модуль широкий функционал, но для базового использования можно обойтись функциями load (чтение данных) и dumps (запись данных).

Запись данных

```
import json


my_data = {
    "a": 15,
    "b": "hello",
    "c": [1, 2, 3]
    }

with open("file.json", "w") as f:
    f.write(json.dumps(my_data))
```

Чтение данных

```
import json


with open("file.json", "r") as f:
    my_data = json.load(f)
    print(my_data)
```

ИСКЛЮЧЕНИЯ

Исключения - это события возникающие в случаях когда интерпретатор языка Python встречается с проблемой которую он не в состоянии решить самостоятельно. Знание основ работы с исключениями в Python является важным навыком т.к. ни одна программа не застрахована от ошибок. Полный список исключений можно найти тут.

```
print(1 / 0)
```

```
Traceback (most recent call last):

  File "C:\Users\Roman\.spyder-py3\temp.py", line 2, in <module>
    print(1 / 0)

ZeroDivisionError: division by zero
```

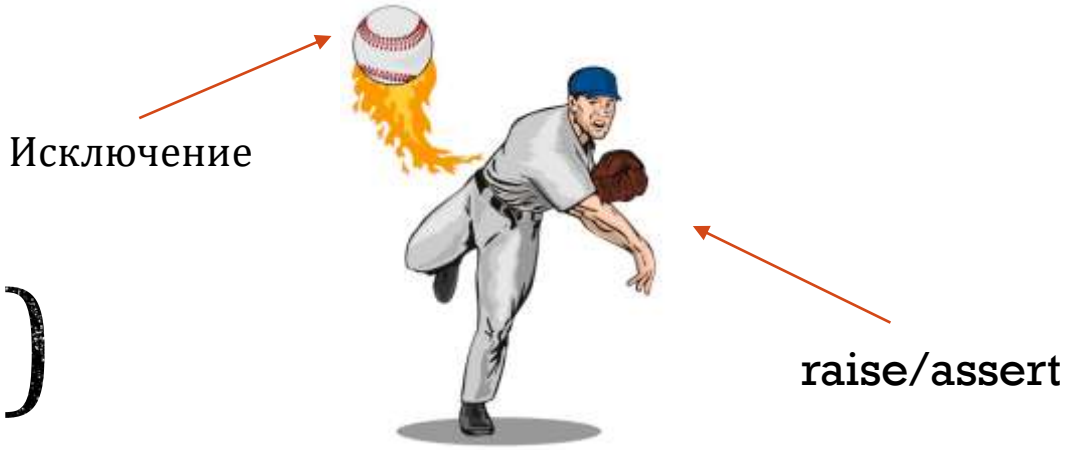
```
print(5 + '4')
```

```
Traceback (most recent call last):

  File "C:\Users\Roman\.spyder-py3\temp.py", line 3, in <module>
    print(5 + '4')

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

ИСКЛЮЧЕНИЯ (ВЫБРАСЫВАНИЕ)



Существует ряд случаев когда возникновение ошибки является желаемым результатом. Например, при проверке начальных условий какого-либо алгоритма, желательным исходом в случае их несоответствия является возникновение ошибки, а не “молчаливое завершение алгоритма”. Существует два ключевых слова позволяющих выбрасывать пользовательские исключение: raise и assert.

```
1 assert 1 == 1
2 assert 2 > 3
```

assert позволяет выбрасывать AssertionError если не выполняется условие указанное после оператора.

```
1 raise Exception("Some text")
```

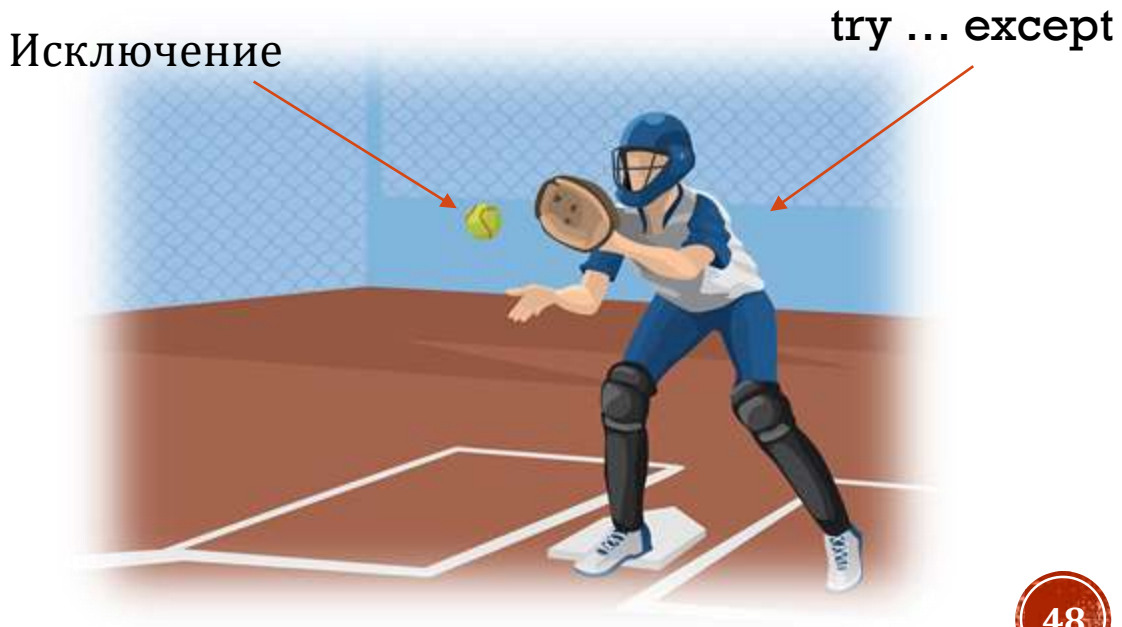
raise позволяет выбрасывать любое исключение, класс которого указывается после оператора

ИСКЛЮЧЕНИЯ (ОБРАБОТКА)

Программа, которая прекращает свое выполнение после первой же ошибки без сообщения конечному пользователю о причинах ее возникновения, является плохой программой Для отлавливания исключений и попытки их программного исправления или формирования отчета существует конструкция try ... except.

Общий вид конструкции

```
try:
    <тело оператора>
except <класс ошибки> as <переменная>:
    <код обработки исключения>
except <класс ошибки>:
    <код обработки исключения>
except:
    <код обработки исключения>
else:
    <когда нет исключения>
finally:
    <выполняется всегда>
```



ИСКЛЮЧЕНИЯ (ОБРАБОТКА)

- Инструкции в этом необязательном блоке выполняются по завершению блока ***try*** без исключений, а также без ***return, continue и break***

```
def ex():
    try:
        return "Норма"
    except:
        return "Ошибка"
    else:
        return "else"
print(ex())  # 'Норма'
```

```
def exr():
    try:
        pass #пустой оператор
    except:
        return "Ошибка"
    else:
        return "else"
print(exr())  # 'else'
```

ИСКЛЮЧЕНИЯ

Блок *finally*

Инструкции из этого блока будут выполнены после выполнения всех прочих блоков, в том числе если исключение не было обработано (в этом случае оно будет выброшено повторно в конце блока ***finally*** автоматически) и если в блоке ***try*** присутствуют ***return*** или ***break***. **При этом информация об исключении недоступна**.

ЗАДАНИЕ ДЛЯ САМОКОНТРОЛЯ

- Задать словарь d равный {'t': -18, 'b': -5, 'h': -18, 'y': 16, 'a': 6}
- Вывести все ключи словаря
- Создать словарь имеющий два новых ключа и один старый (любые ключи, любые значения). Обновить ваш словарь, с помощью нового, используя распаковку.
- Вывести список пар ключ значение.
- Добавить в словарь **d** список под ключом **a** и вывести словарь в файл **json**
- Придумать и реализовать свой пример по использованию исключения

СПАСИБО ЗА ВНИМАНИЕ!