Исключение — поведение отличающееся от ожидаемого. Событие, которое прерывает исполнение программы.

Необходимо понимать когда нам в действительности нужно исключение

```
raise TypeError()
```

```
def find_symbol(where: str, symbol: str) -> int:
  # Это функция, которая ищет вхождение символа в строку и возвращает индекс вхождения
  if not isinstance(where, str) or not isinstance(symbol, str):
    raise AttributeError("То что ищем, и там где ищем должно быть строкой")

  for index, value in enumerate(list(where)):
    if value == symbol:
      return index
  # Что если не нашли, два варианта
  # 1 — поднять исключение [не верный вариант]
  raise AttributeError("Тут такова нету")

  # 2 — вернуть -1 [верный вариант]
  return -1
```

```
>>> lista = [1, 2, 3]
>>> list(enumerate(lista))
[(0, 1), (1, 2), (2, 3)]
```

Мы не используем исключения в легитимном процессе работы программы.
Мы используем исключение для тех случаев, где нам нужно ограничить функционал, предотвратить ошибки.

```
class BananaError(Exception):
  pass

class AttributeError:
  ...

class SuperImportantAttributeError(Exception):
  ...
```

Механизм, который позволяет обрабатывать исключения

try

Обязательно

Любой из

Тут лежит код, который мы пытаемся исполнить

except

Exception-class

as

ex

Блок кода, который будет выполнен в случае возникновения соотвествующего исключения

logger.error(ex)

❗❗ Не пишите в except исключение Exception — ведь тогда мы отсеим все возможные, даже нужные нам исключения

else

Блок кода, который будет выполнен в случае, если исключение не возникло

finally

Блок кода, который будет выполнен в любом случае

```python
dicta = {1: 2, 3: 3, 4: 4}

for key in range(1, 5):
  # 1, 2, 3, 4 — range(1, 5)
  # 1, _  3, 4 — dicta.keys()
  # По идее на двойке мы встрянем
  # но мы хотим перебрать все остальные
  # входящие в ренж ключи
  try:
    print(dicta[key])
  except KeyError as ex:
    print(ex)
  else:
    print("Heh there's a key")
  finally:
    print("Было. Даже если экзепшен — было")
```

with — ключевое слово в python, которое создает контекст для объекта. Следит за тем, чтобы выделенные ресурсы вернулись на свои места.

```python
class Dog:
  def __init__(self, name, length):
    self.name: str = name
    self.length: int = length
    self.is_alive: bool | any = None

  def give_dog_birth(self) -> None:
    self.is_alive = True

  def __len__(self) -> int:
    return self.length

  def __enter__(self) -> None:
    self.give_dog_birth()

  def __exit__(self, err_type, err_value, tb):
    self.is_alive = False

takel = Dog("Biba", 228)

print(len(takel))



with Dog("Biba", 228) as takel:
  print(f"Hello {takel.name}")
```

```python
class Open:
  def __init__(self, path):
    self.file_pointer = None

  def __enter__(self):
    self.file_pointer = open(path, "r")

  def __exit__(self, err_type, err_value, tb):
    self.file_pointer.close()


from pathlib import Path
from pprint import pprint

    # Open.__init__
with Open(Path.home() / "bidla.txt", "r") as fp:
  # Open.__enter__
  pprint(fp.file_pointer.readrows())
# Open.__exit__()

```