



НПГ по КТС гр.Правец

ТЕМА

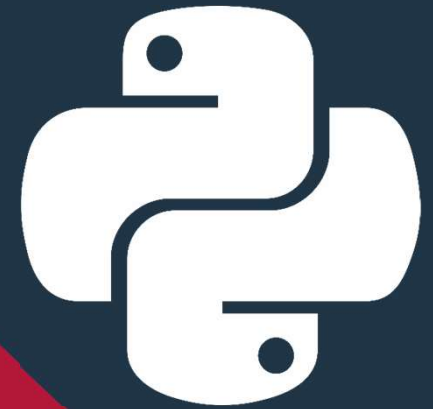
1

Основи на програмирането  
с Python

# TRY EXCEPT ELSE FINALLY

Автор: Любомир Димитров

PYTHON



# Обработка на грешки

- Грешки могат да възникнат по време на изпълнение на програмите. Python предоставя инструменти за обработка на тези грешки.

**ZeroDivisionError**

```
def division_func(a, b):  
    result = a / b  
    return result  
  
num_1 = 2  
num_2 = 0  
  
print(division_func(num_1, num_2))  
  
# ZeroDivisionError
```

# Обработка на грешки

Кой грешки Вие знаете и най-често Ви излизат

Подгответе се да сканирате QR- код с камеритет на вашите телефони



# PYTHON

T-E

try - except

# try - except

- **try** блокът съдържа код, който може да предизвика грешка. Ако грешка възникне, изпълнението на кода в **try** блока се прекъсва, и програмата преминава към **except** блока.
- **except** блокът съдържа код, който се изпълнява само ако в **try** блока възникне грешка.

**ZeroDivisionError**

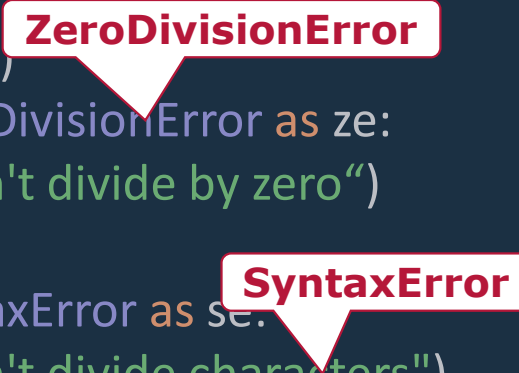
```
try:  
    print(2 / 0)  
except ZeroDivisionError as ze:  
    print("Can't divide by zero")  
# Can't divide by zero
```

# Обработка на няколко грешки

В Python, изключенията са обекти, които се хвърлят, когато възникне грешка. В блока **except**, можем да посочим конкретен тип изключение (например, **ValueError**, **KeyError**) и да изпълним код за обработка, специфичен за всеки вид грешка.

```
try:
    print(2 / a)
except ZeroDivisionError as ze:
    print("Can't divide by zero")

except SyntaxError as se:
    print("Can't divide characters")
# Can't divide characters
```



# PYTHON

el

else:

# В краен случай else:

**else** блокът съдържа код, който се изпълнява само ако в **try** блока няма възникнала грешка. Това предоставя възможност за изпълнение на алтернативен код, когато няма проблеми.

```
try:
    print(6 / 3)
except ZeroDivisionError as ze:
    print("Can't divide by zero")
except SyntaxError as se:
    print("Can't divide characters")
else:
    print("Divided by integer")
# 2.0
# Divide by Integer
```



# PYTHON

f

finally:

# Try-except-finally - блок

**finally** блокът съдържа код, който се изпълнява винаги, независимо дали в **try** блока има грешка или не. Това е полезно за освобождаване на ресурси или изпълнение на код, който трябва да се изпълни във всеки случай.

```
try:
    print(6 / 3)
except ZeroDivisionError as ze:
    print("Can't divide by zero")
except SyntaxError as se:
    print("Can't divide characters")
finally:
    print("That is my calculations")
# 2.0
# That is my calculations
```

```
try:
    print(6 / a)
except ZeroDivisionError as ze:
    print("Can't divide by zero")
except SyntaxError as se:
    print("Can't divide characters")
finally:
    print("That is my calculations")
# Can't divide characters
# That is my calculations
```

```
try:
    print(6 / 0)
except ZeroDivisionError as ze:
    print("Can't divide by zero")
except SyntaxError as se:
    print("Can't divide characters")
finally:
    print("That is my calculations")
# Can't divide by zero
# That is my calculations
```

# Множество except блокове

Подобно на **if-elif-else** където можем да имаме множество **elif** проверки. Така и тук можем да имаме повече от един **except** в един **try-except-else-finally** блок. Множеството **except** блокове позволяват на програмиста да се справи с различни видове грешки по индивидуален начин. Първият подходящ блок се изпълнява, и изпълнението продължава оттам.

```
try:
    print(6 / 3)
except ZeroDivisionError as ze:
    print("Can't divide by zero")
except SyntaxError as se:
    print("Can't divide characters")
except Exception as e:
    print("Something went wrong")
.....
```

```
...
except SystemError as sye:
    print("Something went wrong again")
else:
    print("Everything works")
finally:
    print("That is my calculations")
```

# Вградени изключения в Python

Python предоставя богата библиотека от вградени изключения, които се използват за представяне на различни видове грешки. Някои от тях включват:

`TypeError`

`ValueError`

`NameError`

`IndexError`

`KeyError`

`SyntaxError`

# Полезни съвети при обработка на грешки

---

- Подчертайте важността на конкретност при обработка на грешки, където е възможно. Това помага за по-добра идентификация и отстраняване на проблемите.
- Избягвайте широки блокове **try-except**, за да избегнете нежелано скриване на грешки и трудности в откриването на проблеми.
- Препоръчително е четенето на документацията за вградените изключения, за да се разбере какви грешки могат да възникнат.

# Finally 😊

---

Обработката на грешки в Python е неотделима част от разработването на стабилни и надеждни приложения. Презентацията ни демонстрира ключовите концепции - **try**, **except**, **else** и **finally** - които програмистите използват за ефективно справяне с възможни проблеми по време на изпълнение на кода.

Завършваме с акцент върху важността на предотвратяването на необработени грешки. Необработените грешки могат да доведат до непредвидени състояния на програмата и да предизвикат нежелани последствия. Затова е от съществено значение да се постави ударение върху коректната и изчерпателна обработка на грешки в програмния код.

Този заключителен слайд обобщава не само ключовите теми от презентацията, но и подчертава значението на добрата практика при обработката на грешки в програмирането.



НПГ по КТС гр.Правец

ТЕМА | 1

---

# БЛАГОДАРЯ ЗА ВНИМАНИЕТО

Автор: Любомир Димитров

PYTHON

