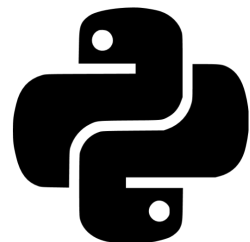


ThoughtWorks®

não precisa importar

PYTHON BUILTINS

Um passeio pelas funções e classes embutidas no Python 3.7+



JUST PYTHON
release v3.0
2019-11-09

Luciano Ramalho
@ramalhoorg

ThoughtWorks®

HELP

É sempre bom poder pedir ajuda

PRIMEIRA OLHADA: DIR E HELP

```
Python 3.8.0 (v3.8.0:fa919fdf25, Oct 14 2019, 10:23:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__',
 '__spec__']
>>> help(__builtins__)
```

Help on built-in module builtins:

NAME

builtins – Built-in functions, exceptions, and other objects.

DESCRIPTION

Noteworthy: None is the `nil' object; Ellipsis represents `...' in slices.

CLASSES

object

 BaseException

 Exception

 ArithmeticError

 FloatingPointError

 OverflowError

 ZeroDivisionError

 AssertionError

 AttributeError

 BufferError

**8618 linhas
de ajuda**

LISTA HIERÁRQUICA DE CLASSES NO HELP

```
UnicodeEncodeError
UnicodeTranslateError
Warning
    BytesWarning
    DeprecationWarning
    FutureWarning
    ImportWarning
    PendingDeprecationWarning
    ResourceWarning
    RuntimeWarning
    SyntaxWarning
    UnicodeWarning
    UserWarning
GeneratorExit
KeyboardInterrupt
SystemExit
bytearray
bytes
classmethod
complex
dict
enumerate
filter
float
frozenset
int
    bool
list
```

Um monte de
exceções...

Classes com
nomes em caixa
baixa



bool é
subclasse de
int!



USANDO PYTHON PARA APRENDER PYTHON

Como é bom ter um console interativo

LISTA DE NOMES EM `__BUILTINS__`

```
>>> print(*dir(__builtins__), sep=' ')
ArithmeticError AssertionError AttributeError BaseException BlockingIOError
BrokenPipeError BufferError BytesWarning ChildProcessError
ConnectionAbortedError ConnectionError ConnectionRefusedError
ConnectionResetError DeprecationWarning EOFError Ellipsis EnvironmentError
Exception False FileNotFoundError FloatingPointError
FutureWarning GeneratorExit IOError ImportError ImportWarning
IndentationError IndexError InterruptedError IsADirectoryError KeyError
KeyboardInterrupt LookupError MemoryError ModuleNotFoundError NameError None
NotADirectoryError NotImplemented NotImplementedError OSError OverflowError
PendingDeprecationWarning PermissionError ProcessLookupError RecursionError
ReferenceError ResourceWarning RuntimeError RuntimeWarning StopAsyncIteration
StopIteration SyntaxError SyntaxWarning SystemError SystemExit TabError
TimeoutError True TypeError UnboundLocalError UnicodeDecodeError
UnicodeEncodeError UnicodeError UnicodeTranslateError UnicodeWarning
UserWarning ValueError Warning ZeroDivisionError _ __build_class__ __debug__
__doc__ __import__ __loader__ __name__ __package__ __spec__ abs all any
ascii bin bool breakpoint bytearray bytes callable chr classmethod
compile complex copyright credits delattr dict dir divmod enumerate eval
exec exit filter float format frozenset getattr globals hasattr hash
help hex id input int isinstance issubclass iter len license list
locals map max memoryview min next object oct open ord pow print
property quit range repr reversed round set setattr slice sorted
staticmethod str sum super tuple type vars zip
```

ÁRVORE DE EXCEÇÕES

Ou: como listar uma hierarquia de classes

LISTA DE NOMES EM CAIXA-MISTA, SÓ INVOCÁVEIS

```
>>> Names = [s for s in dir(__builtins__) if s.lower() != s]
>>> Exceptions = [s for s in Names if callable(getattr(__builtins__, s))]
```

>>> print(*Exceptions, sep=' ')

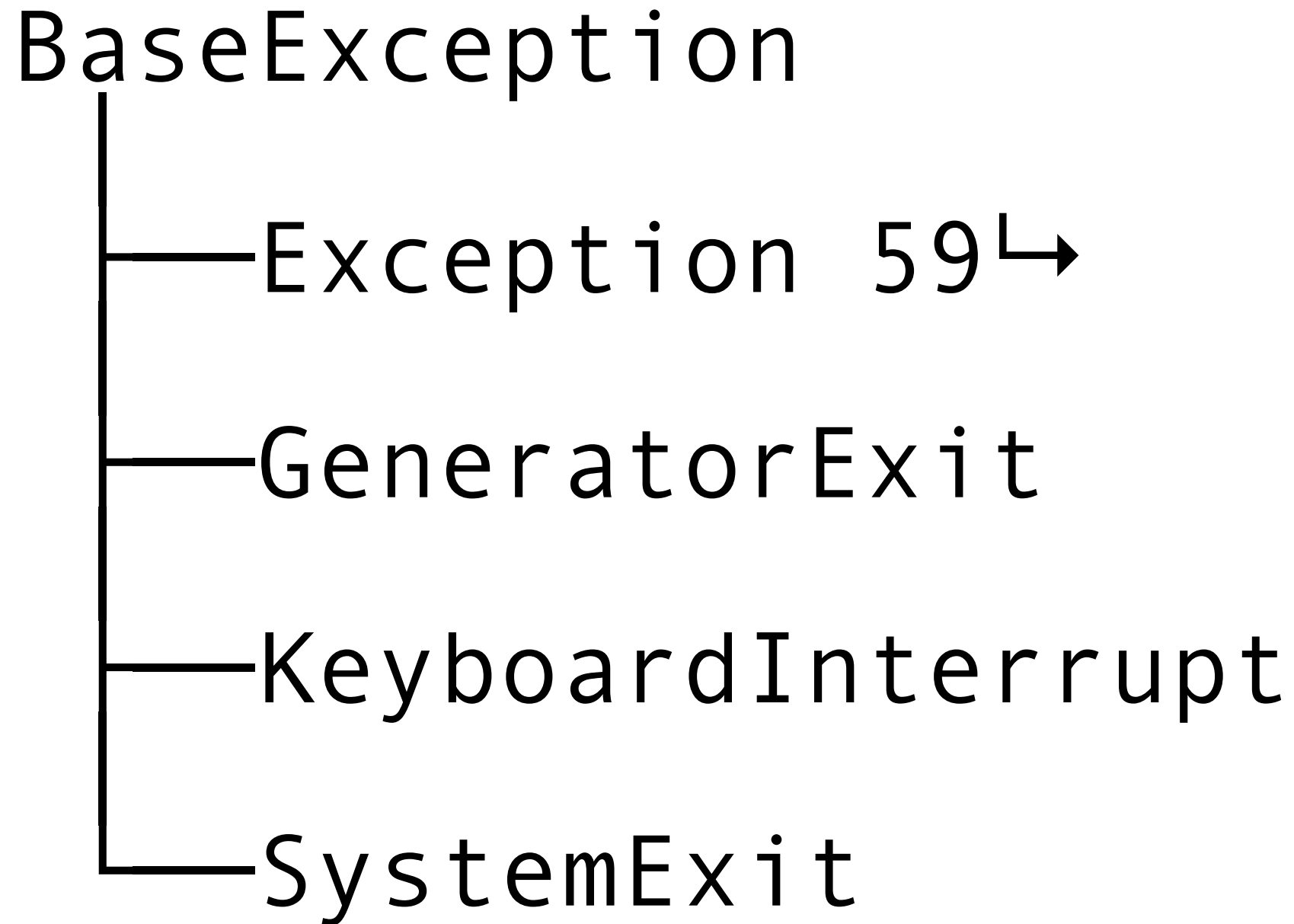
ArithmeticError AssertionError AttributeError BaseException
BlockingIOError BrokenPipeError BufferError BytesWarning
ChildProcessError ConnectionAbortedError ConnectionError
ConnectionRefusedError ConnectionResetError DeprecationWarning EOFError
EnvironmentError Exception FileExistsError FileNotFoundError
FloatingPointError FutureWarning GeneratorExit IOError ImportError
ImportWarning IndentationError IndexError InterruptedError
IsADirectoryError KeyError KeyboardInterrupt LookupError MemoryError
ModuleNotFoundError NameError NotADirectoryError NotImplemented
OSError OverflowError PendingDeprecationWarning PermissionError
ProcessLookupError RecursionError ReferenceError ResourceWarning
RuntimeError RuntimeWarning StopAsyncIteration StopIteration
SyntaxError SyntaxWarning SystemError SystemExit TabError
TimeoutError TypeError UnboundLocalError UnicodeDecodeError
UnicodeEncodeError UnicodeError UnicodeTranslateError UnicodeWarning
UserWarning ValueError Warning ZeroDivisionError

Notebook com códigos para explorar `__builtins__`: <http://bit.ly/2Q5x3oa>

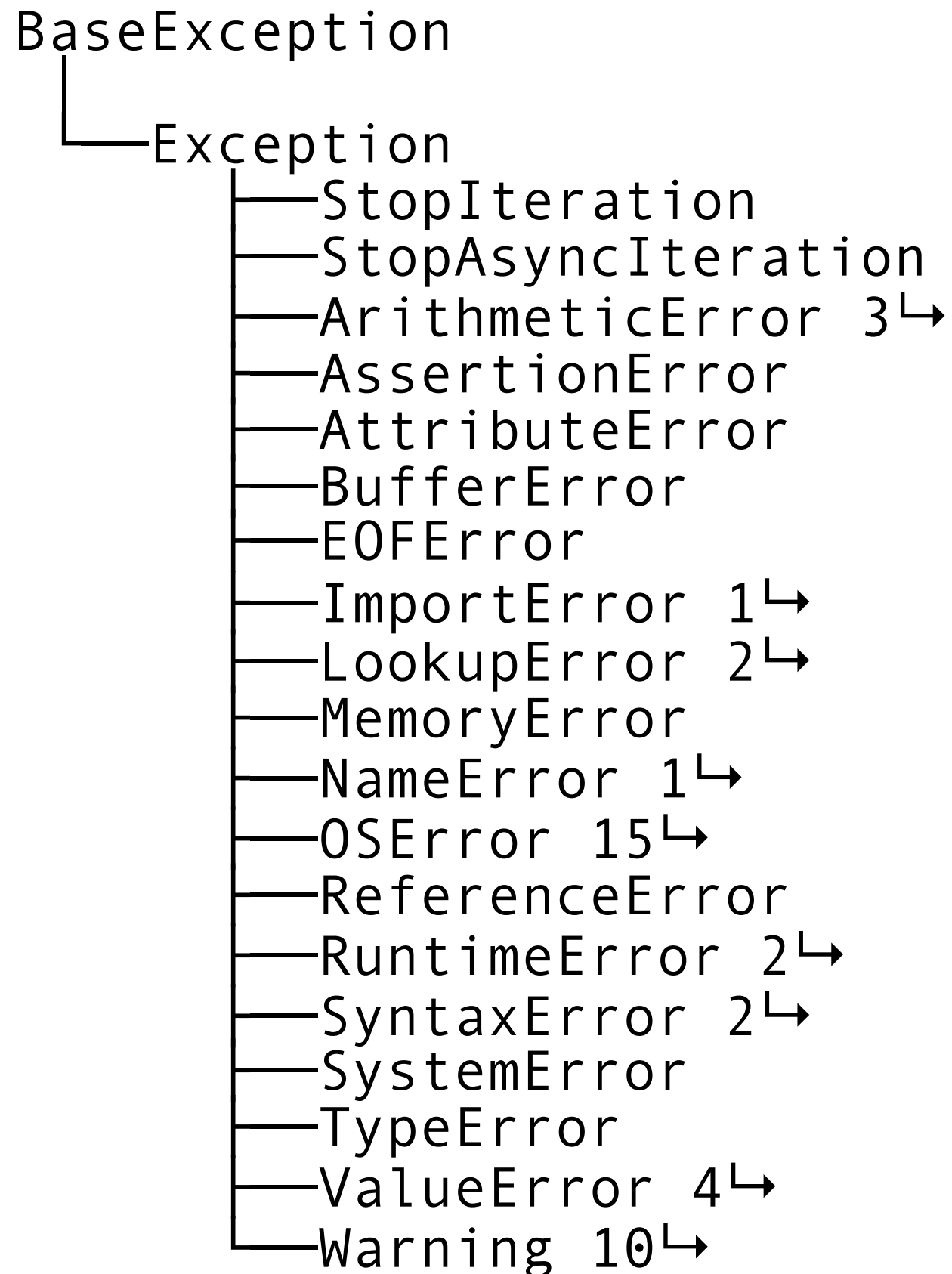
HIERARQUIA DE EXCEÇÕES

```
>>> def tree(cls, level=0):
...     yield ' ' * 4 * level + cls.__name__
...     for sub in sorted(cls.__subclasses__(), key=lambda c: c.__name__):
...         yield from tree(sub, level+1)
...
>>> print(*tree(BaseException), sep='\n')
BaseException
    Exception
        ArithmeticError
            FloatingPointError
            OverflowError
            ZeroDivisionError
        AssertionError
        AttributeError
        BufferError
        EOFError
        EndOfBlock
        Error
        ErrorDuringImport
        ImportError
            ModuleNotFoundError
            ZipImportError
        LookupError
            CodecRegistryError
            IndexError
```

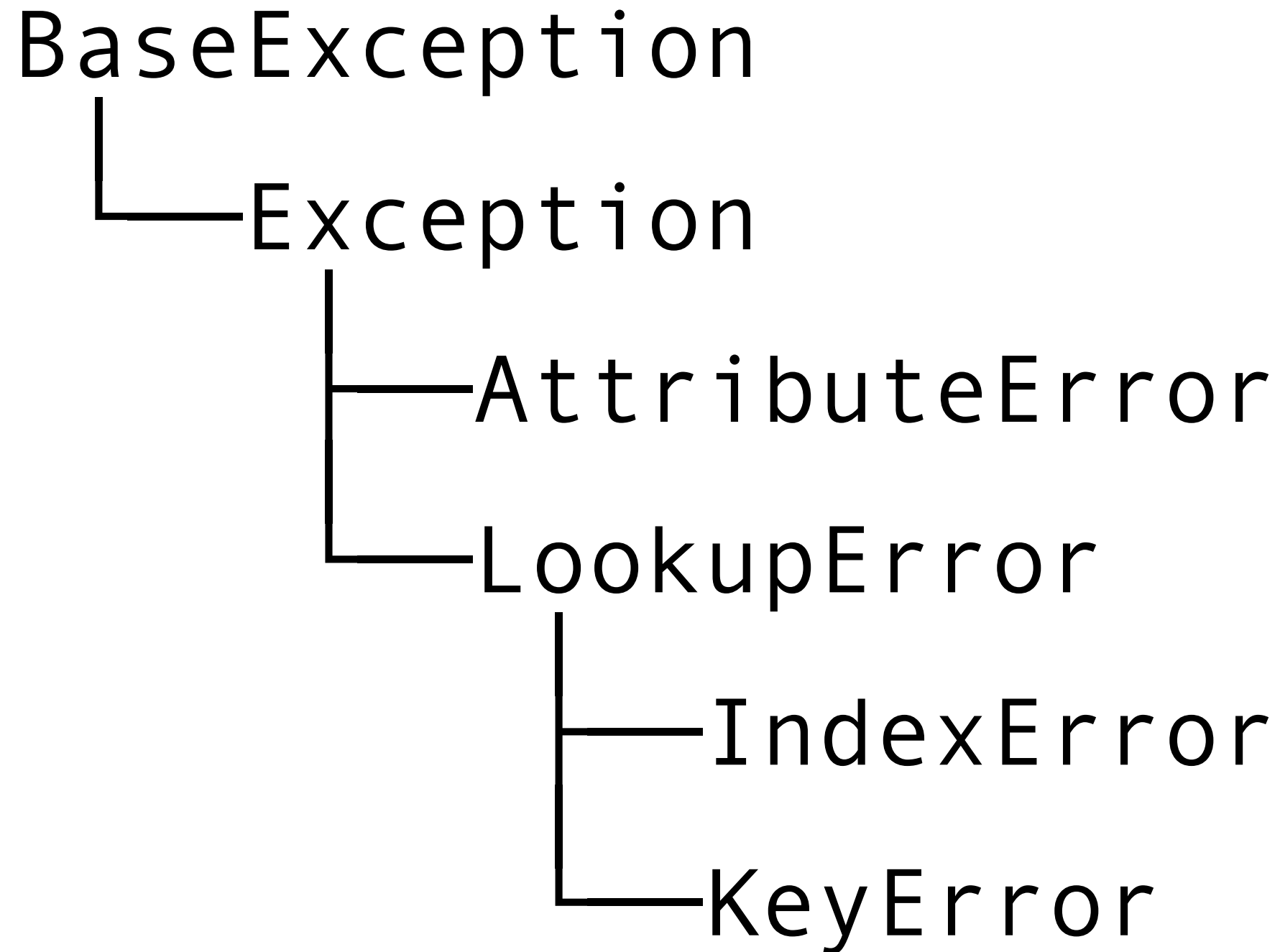
TOPO DA HIERARQUIA DE EXCEÇÕES



SUBCLASSES DIRETAS DE EXCEPTION



ERROS DE ACESSO A ATRIBUTOS OU ITENS



ERROS DE UNICODE

BaseException

└─ Exception

└─ ValueError

└─ UnicodeError

└─ UnicodeDecodeError

└─ UnicodeEncodeError

└─ UnicodeTranslateError

OUTROS EMBUTIDOS

A melhor parte dos built-ins: funções e classes

LISTA DE NOMES EM CAIXA-BAIXA SEM PREFIXO _

```
>>> names = [s for s in dir(__builtins__) if s == s.lower() and s[0] != '_']
>>> print(*names, sep=' ')
abs all any ascii bin bool breakpoint bytearray bytes callable chr
classmethod compile complex copyright credits delattr dict dir divmod
enumerate eval exec exit filter float format frozenset getattr
globals hasattr hash help hex id input int isinstance issubclass
iter len license list locals map max memoryview min next object oct
open ord pow print property quit range repr reversed round set
setattr slice sorted staticmethod str sum super tuple type vars zip
```

MESMA LISTA DE ANTES, AGORA COM TIPOS

```
>>> categorized = []
>>> for name in names:
...     obj = getattr(__builtins__, name)
...     categorized.append((type(obj).__name__, name))
...
>>> print(*categorized, sep='\n')
('builtin_function_or_method', 'abs')
('builtin_function_or_method', 'all')
('builtin_function_or_method', 'any')
('builtin_function_or_method', 'ascii')
('builtin_function_or_method', 'bin')
('type', 'bool')
('builtin_function_or_method', 'breakpoint')
('type', 'bytearray')
('type', 'bytes')
('builtin_function_or_method', 'callable')
('builtin_function_or_method', 'chr')
('type', 'classmethod')
('builtin_function_or_method', 'compile')
('type', 'complex')
('_Printer', 'copyright')
('_Printer', 'credits')
('builtin_function_or_method', 'delattr')
('type', 'dict')
('builtin_function_or_method', 'dir')
```


MESMA LISTA DE ANTES, AGRUPADA POR TIPOS

```
>>> for category, group in groupby(sorted(categorized), itemgetter(0)):
...     print('-' * 20, category)
...     for _, name in group:
...         print(name)
...
_____ Quitter
exit
quit
_____ _Helper
help
_____ _Printer
copyright
credits
license
_____ builtin_function_or_method
abs
all
any
ascii
bin
breakpoint
callable
chr
compile
delattr
```

FUNÇÕES

CLASSES

abs	format	min	bool	memoryview
all	getattr	next	bytearray	object
any	globals	oct	bytes	property
ascii	hasattr	open	classmethod	range
bin	hash	ord	complex	reversed
breakpoint	hex	pow	dict	set
callable	id	print	enumerate	slice
chr	input	repr	filter	staticmethod
compile	isinstance	round	float	str
delattr	issubclass	setattr	frozenset	super
dir	iter	sorted	int	tuple
divmod	len	sum	list	type
eval	locals	vars	map	zip
exec	max			

Essa distinção não é importante na prática...

LISTA DE NOMES EM CAIXA-MISTA, NÃO INVOCÁVEIS

```
>>> Const = [s for s in Names if not callable(getattr(__builtins__, s))]  
>>> print(*Const, sep=' ')  
Ellipsis False None NotImplemented True
```

LISTA DE NOMES EM CAIXA-MISTA, NÃO INVOCÁVEIS

```
>>> Const = [s for s in Names if not callable(getattr(__builtins__, s))]  
>>> print(*Const, sep=' ')  
Ellipsis False None NotImplemented True
```

Conferindo que os nomes acima são todos os nomes em caixa-mista que não são classes de exceções:

```
>>> set(Names) - set(Exceptions)  
{'Ellipsis', 'NotImplemented', 'False', 'None', 'True'}
```

INVOCÁVEIS BUILT-IN SEPARADOS EM CATEGORIAS

FUNÇÕES BÁSICAS

exibição de objetos

ascii
format
repr
str*

exibição de inteiros

bin
hex
oct

entrada/saída

input
open
print

unicode

chr
ord

matemática

abs
divmod
pow
round

TIPOS DE DADOS

tipos simples

bool
complex
float
int
object

sequências planas

bytearray
bytes
memoryview
str*

containers

dict
frozenset
list
set
tuple

PROCESSAMENTO

operações com coleções

all
any
len
max
min
sorted
sum

iteradores/geradores

enumerate
filter
map
range
reversed
zip

iteração em baixo nível

iter
next

FERRAMENTAS DE PROGRAMAÇÃO

decoradores de métodos

classmethod
property
staticmethod

delegação dinâmica

super

introspecção de objetos

callable
dir*
hash
id
isinstance
issubclass
type*

singletons especiais

Ellipsis
False
None
NotImplemented
True

operações com atributos

delattr
getattr
hasattr
setattr

inspeção do ambiente

breakpoint
dir*
globals
locals
vars

utilitárias

compile
eval
exec
slice
type*
__import__

Categorização subjetiva, como toda categorização

TIPOS DE DADOS

Alguns detalhes pode surpreender

BOOLE INT

Tamanho limitado só pela RAM

Conversão nos dois sentidos

Todo bool é um int

USANDO BOOL COMO INT

```
>>> def comissao(vendas, bonus=500):  
...     if vendas > 10_000:  
...         return vendas * 0.1 + bonus  
...     else:  
...         return vendas * 0.1
```

```
>>> comissao(20_000)
```

```
2500.0
```

```
>>> comissao(500)
```

```
50.0
```

```
>>> def comissao(vendas, bonus=500):  
...     return vendas * 0.1 + (vendas > 10_000) * bonus
```

```
>>> comissao(20_000)
```

```
2500.0
```

```
>>> comissao(500)
```

```
50.0
```

**Use com
moderação**

OBJECT

Principais usos:

- Superclasse padrão (implícita no Python 3)
- Sentinela: valor único no sistema, para sinalizar o fim de alguma série de dados ou terminar processo (*"poison pil"*)

```
>>> marcador1 = object()
>>> marcador1
<object object at 0x1025c2ff0>
>>> marcador2 = object()
>>> marcador2
<object object at 0x1025c2ec0>
>>> marcador1 is marcador2
False
>>> marcador1 == marcador2
False
```

FLOAT E COMPLEX

float: padrão IEEE 754 double (64 bits)

complex: dois floats

float

```
>>> x = 1.1
>>> y = 0.11
>>> z = 0.011
>>> y * 10 == x
True
>>> z * 100 == x
False
>>> f'{x:0.20f}, {y*10:0.20f}, {z*100:0.20f}'
'1.100000000000000000008882, 1.100000000000000000008882, 1.0999999999999999986677'
>>> math.isclose(x, z*100)
True
```

complex

```
>>> w = 3 + 4j
>>> w
(3+4j)
>>> w.real
3.0
>>> w.imag
4.0
>>> abs(w)
5.0
```

math.isclose(): o jeito certo de comparar floats!

Notebook demonstrando alguns tipos de dados: <http://bit.ly/33B9rLQ>

PROCESSAMENTO DE DADOS

Tratamento eficiente de coleções

ITER

Uso alternativo de **iter**: com segundo argumento, cria iterador que invoca função até que um valor sentinela apareça.

```
>>> from random import randint
>>> def d6():
...     return randint(1, 6)
...
>>> [d6() for _ in range(20)]
[5, 5, 4, 3, 4, 5, 2, 4, 3, 3,
6, 6, 5, 1, 6, 5, 2, 3, 1, 2]
```

```
>>> for lance in iter(d6, 1):
...     print(lance)
...
2
6
>>> for lance in iter(d6, 1):
...     print(lance)
...
2
2
4
3
>>> for lance in iter(d6, 1):
...     print(lance)
...
2
4
3
3
6
2
```

PROCESSAMENTO DE MASSAS DE DADOS

sorted, max, min: com critérios de ordenação customizável

sum, all, any: funções que reduzem iterável a um único valor

enumerate, zip, map, filter: geradores que consomem iteráveis

Notebook demonstrando funções e classes citadas acima: <http://bit.ly/33JGaip>

O QUE FALTOU...

Muita coisa!

LIÇÃO APRENDIDA: BUILTINS NÃO CABEM EM 1 AULA!

FUNÇÕES BÁSICAS

exibição de objetos

ascii
format
repr
str*

exibição de inteiros

bin
hex
oct

entrada/saída

input
open
print

unicode

chr
ord

matemática

abs
divmod
pow
round

TIPOS DE DADOS

tipos simples

bool
complex
float
int
object

sequências planas

bytearray
bytes
memoryview
str*

containers

dict
frozenset
list
set
tuple

PROCESSAMENTO

operações com coleções

all
any
len
max
min
sorted
sum

iteradores/geradores

enumerate
filter
map
range
reversed
zip

iteração em baixo nível

iter
next

FERRAMENTAS DE PROGRAMAÇÃO

decoradores de métodos

classmethod
property
staticmethod

delegação dinâmica

super

introspecção de objetos

callable
dir*
hash
id
isinstance
issubclass
type*

singletons especiais

Ellipsis
False
None
NotImplemented
True

operações com atributos

delattr
getattr
hasattr
setattr

inspeção do ambiente

breakpoint
dir*
globals
locals
vars

utilitárias

compile
eval
exec
slice
type*
__import__

MUITO GRATO!

Luciano Ramalho
@ramalhoorg

lramalho@thoughtworks.com

ThoughtWorks®