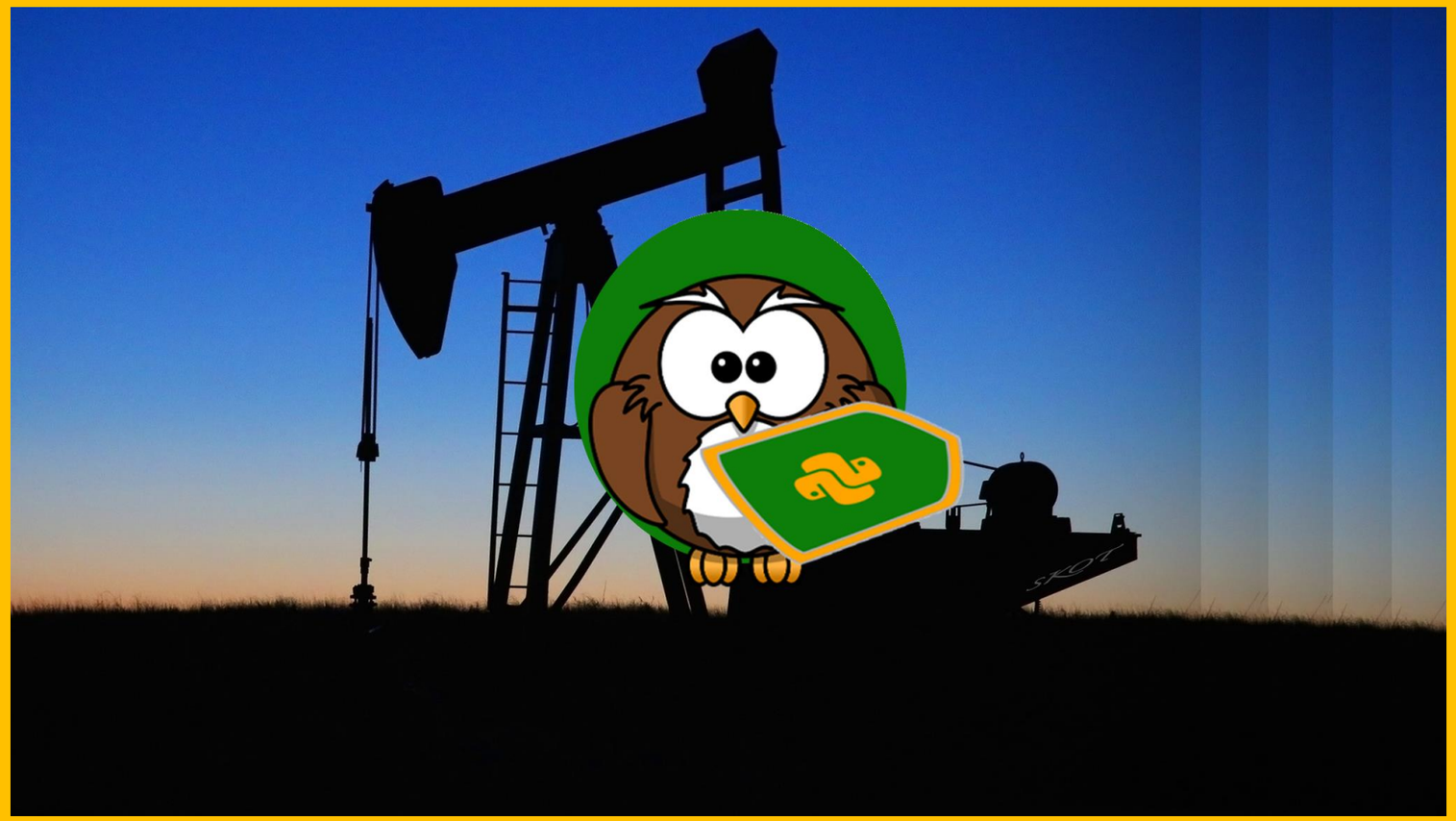
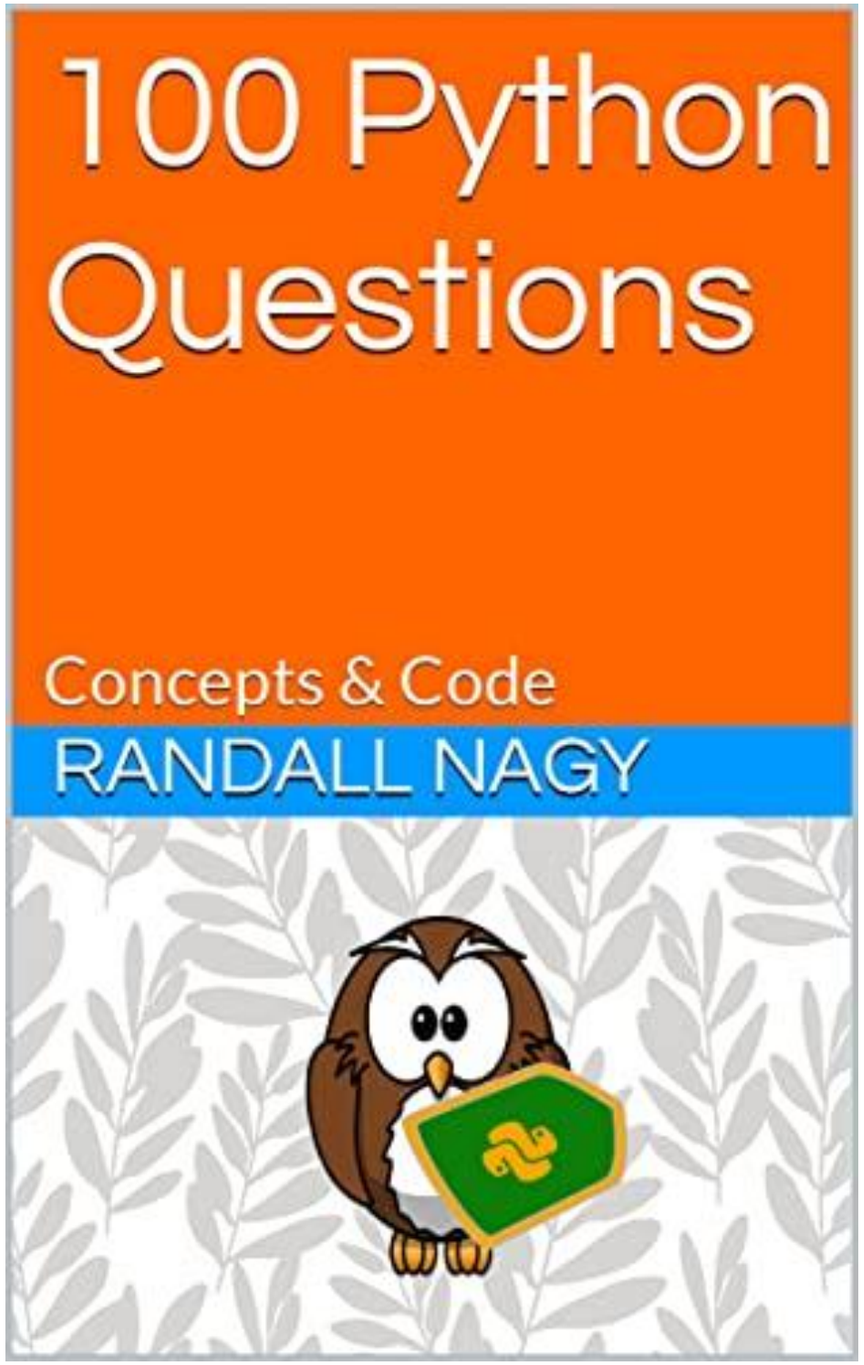
Modern Pythoneering

The Built-In Reports

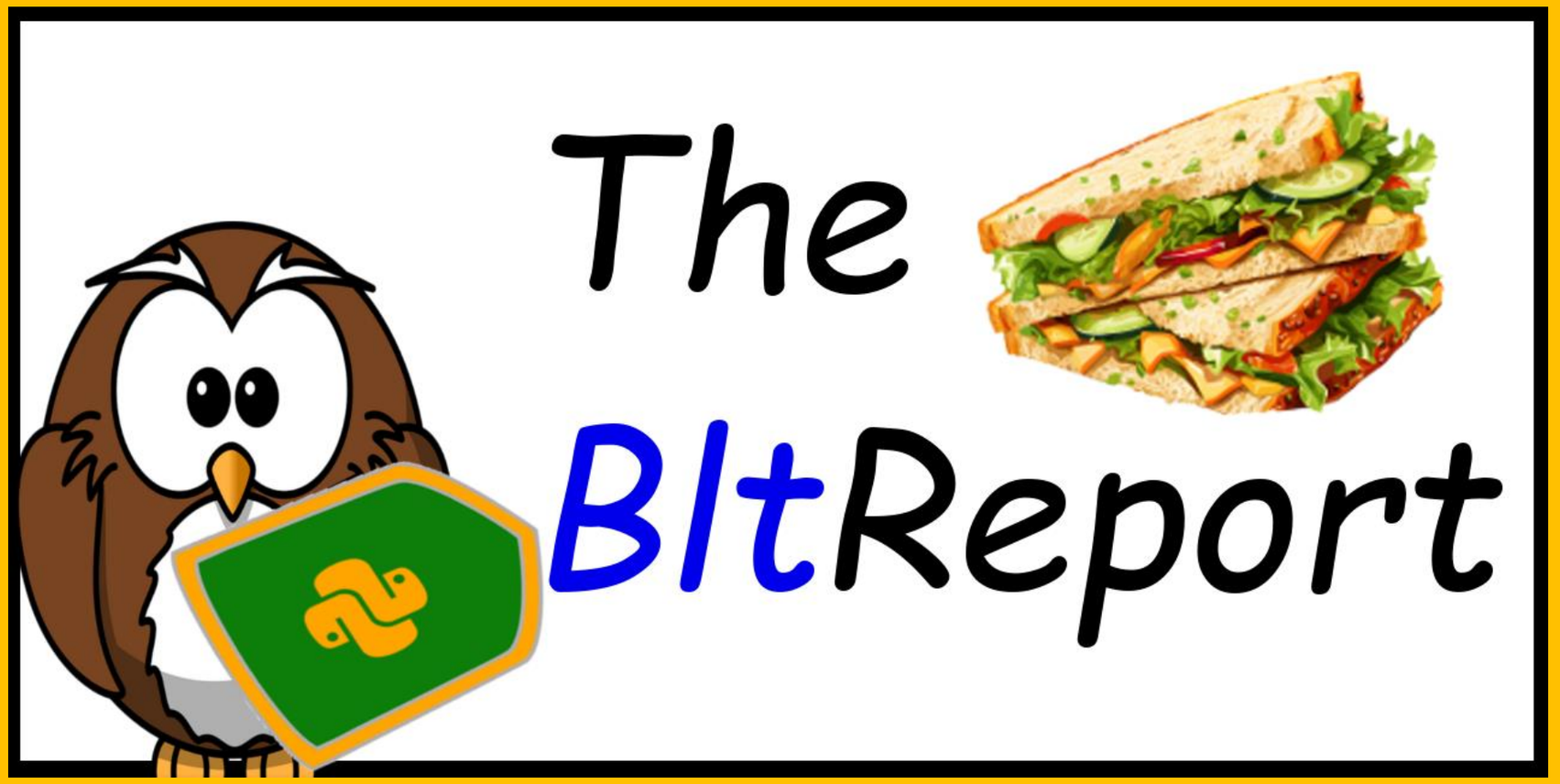
By Randall Nagy

A.K.A: PyQuesting!




100 Python Questions
Concepts & Code
RANDALL NAGY

Video: BLT_00100



The ‘Upper-Cased’

Keywords:

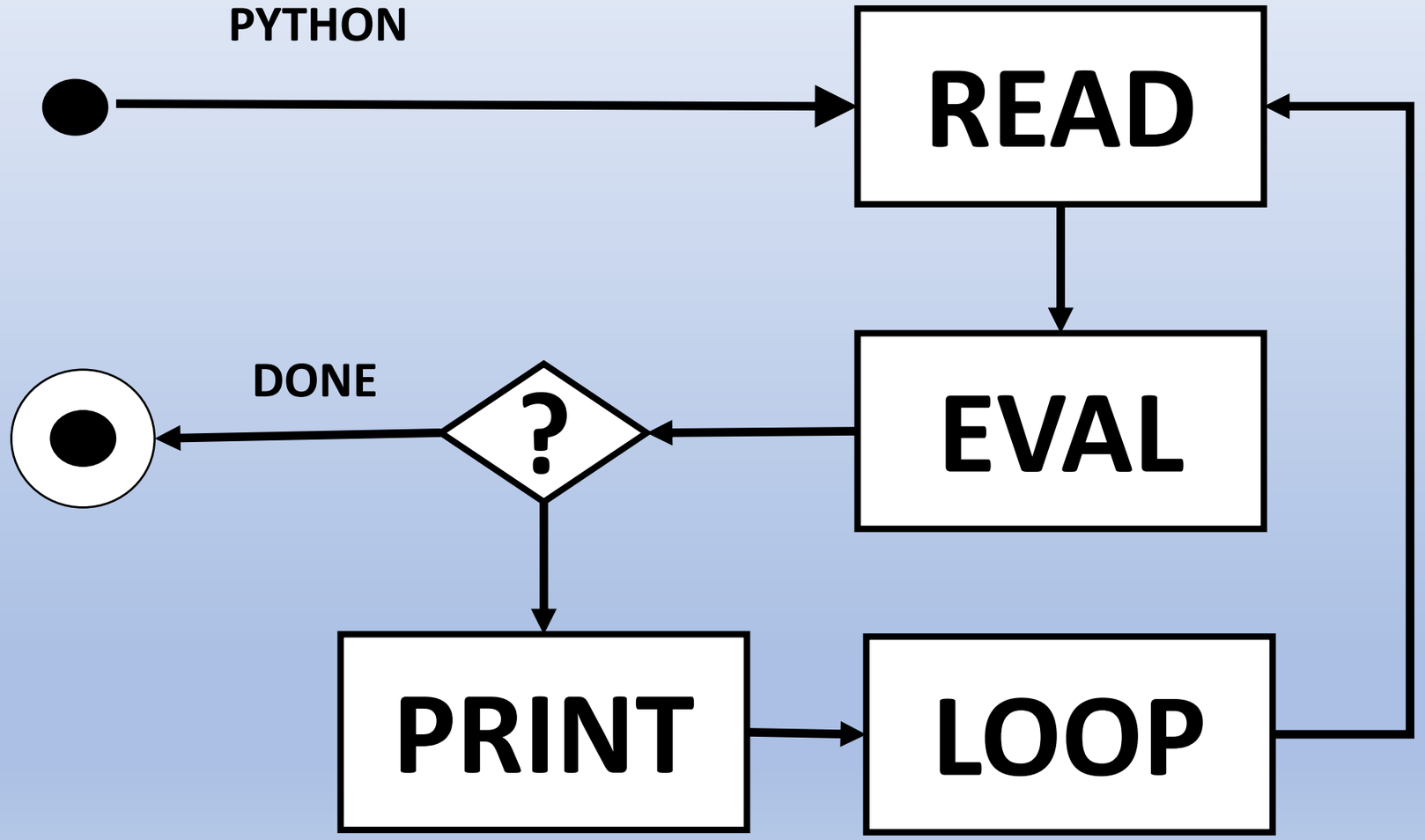
- True
- False
- None

The Built-Ins

Ops:

- type()
- eval()
- bool()
- int()

R.E.P.L ?



KA1002: The REPL

Beginner

What is REPL?

(1) All objects are REPLacable
(2) Read, Evaluate, Print, and Loop
(3) The default version of Python
(4) A well-known research & design pattern
(5) None of the above

KA1056: Boolean Basics

Beginner

Boolean Values:

(1) Either `True` or `False`
(2) Can include `None`
(3) Are default return types
(4) May be lower cased
(5) All of the above

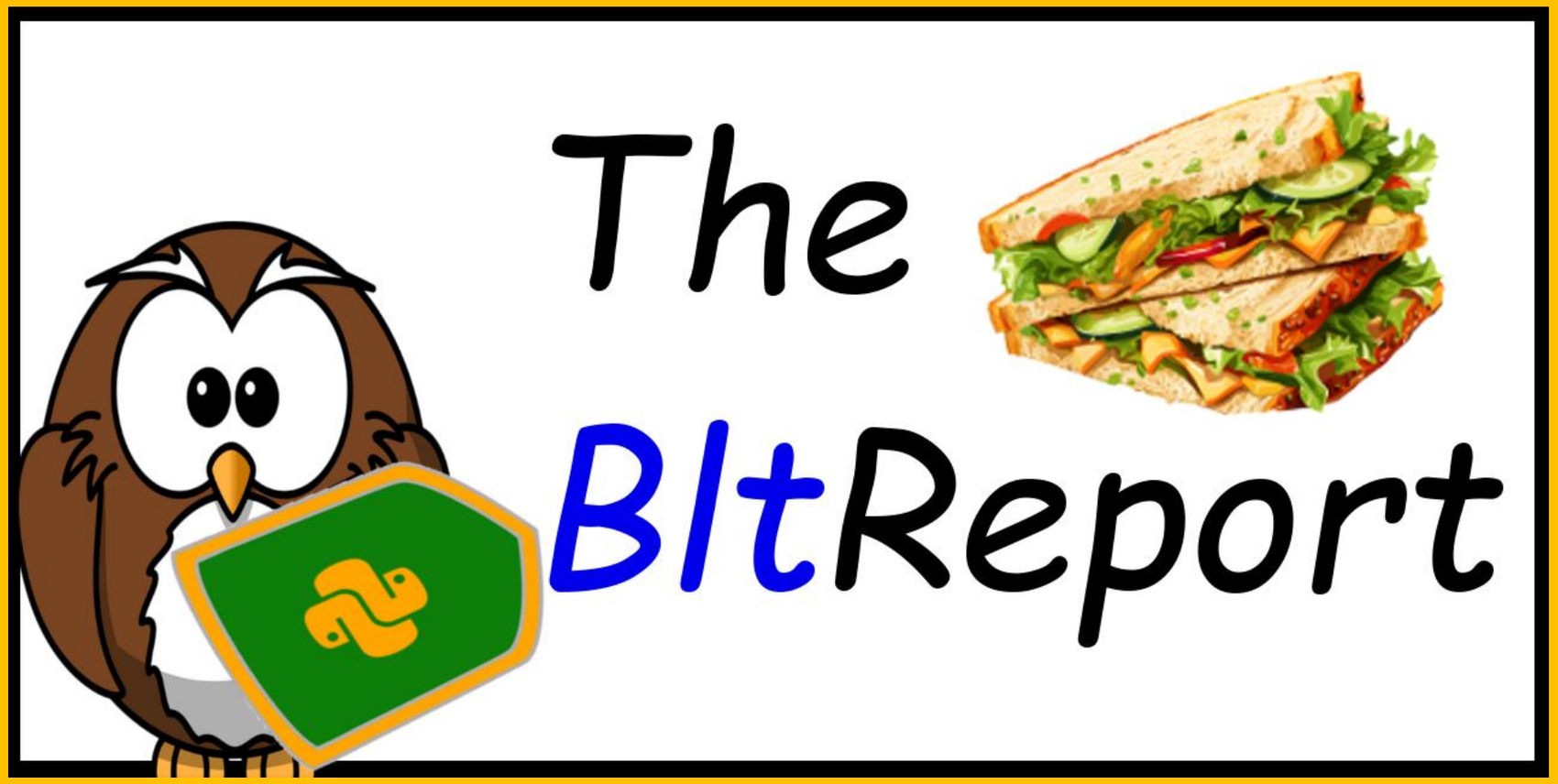
KA1060: Evaluations

Beginner

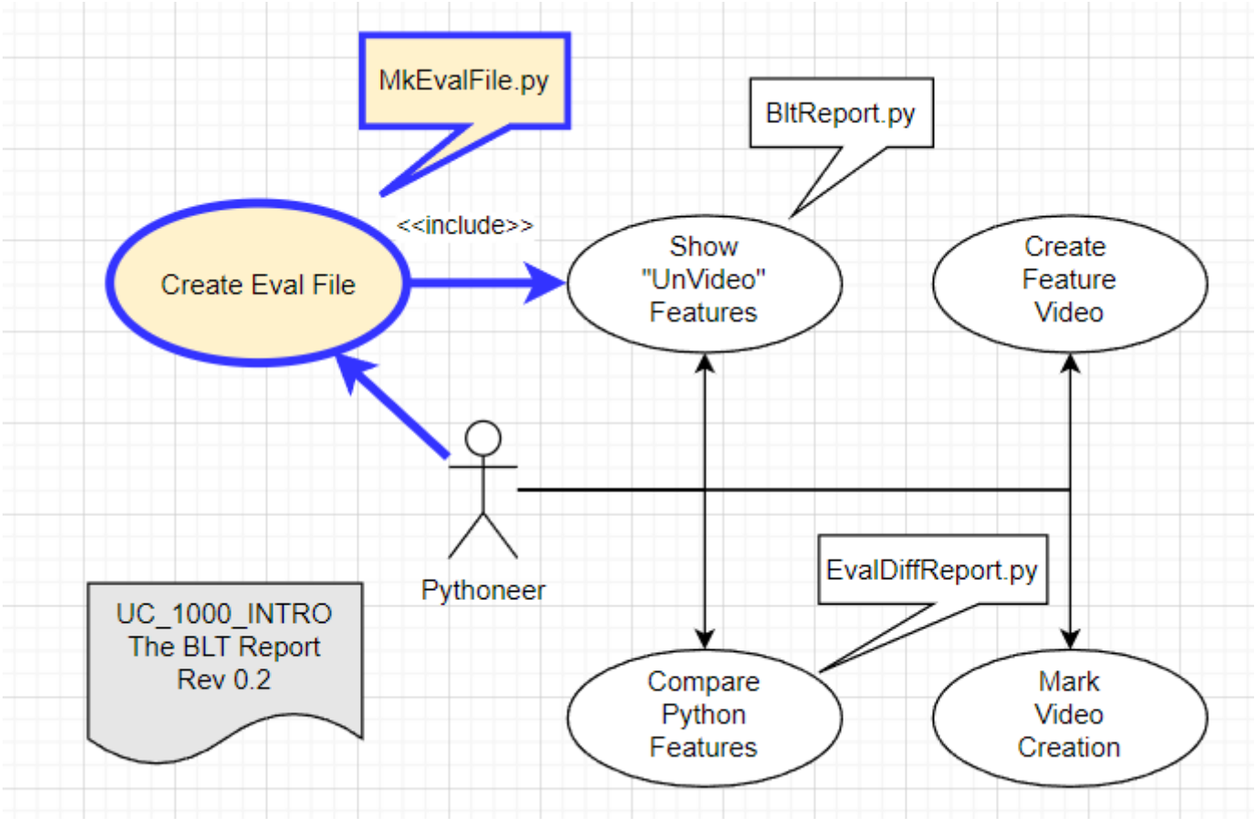
```
eval('bool(1)') will:
```

(1) Raise an Exception
(2) Return True
(3) Return False
(4) Return NoneType
(5) None of the above

Video: BLT_00200



Code Changes



Review

Ops:

- print()
- int()
- bool()
- type()
- eval() …

Reviewing print() Options

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Common int() / bool() Members

- Comprehension

```
>>> print(*[z for z in dir(7) if not z[0] == '_'],sep='\n')
as_integer_ratio
bit_length
conjugate
denominator
from_bytes
imag
numerator
real
to_bytes
>>>
```

.bit_length?

- bool() ~v~ int()

```
>>> bool(1).bit_length()
1
>>> int(1).bit_length()
1
>>> int(255).bit_length()
8
>>> bool(255).bit_length()
1
```

From / To Bytes

```
>>> int(1).to_bytes(2,'big',signed=False)
b'\x00\x01'
>>> int(1).to_bytes(2,'little',signed=False)
b'\x01\x00'
>>> int().from_bytes(b'\x01\x00','little',signed=False)
1
```

```
>>> little = int(1).to_bytes(2,'little',signed=False)
>>> print(little)
b'\x01\x00'
>>> print(int().from_bytes(little,'big',signed=False))
256
```

Conjugate

```
>>> import math
>>> i = int(math.pi)
>>> i.conjugate()
3
>>>
>>> i = int(-math.pi)
>>> i.conjugate()
-3
```

as_integer_ratio

```
>>> help(int(7).as_integer_ratio)
Help on built-in function as_integer_ratio:

as_integer_ratio() method of builtins.int instance
    Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the
original int and with a positive denominator.
```

Numerator : Ratios

```
>>> int(7).as_integer_ratio()
(7, 1)
>>> int(-7).as_integer_ratio()
(-7, 1)
>>> int(0xe7).as_integer_ratio()
(231, 1)
>>> int(-0xe7).as_integer_ratio()
(-231, 1)
```

Properties

```
>>> int(7).numerator
7
>>> int(7).denominator
1
```

Properties

```
>>> import math
>>> math.pi
3.141592653589793
>>> int(math.pi)
3
>>> i = int(math.pi)
>>> i.imag
0
>>> i.real
3
```

Integral Type Commons

- ✓as_integer_ratio
- ✓bit_length
- ✓conjugate
- ✓denominator
- ✓from_bytes
- ✓imag
- ✓numerator
- ✓real
- ✓to_bytes

Class Dictionaries

- __dict__ ~v~ vars()

```
>>> class Z:
        a=1;b=2
        def __init__(self):
            self.c=7;self.d=8


>>> Z().__dict__
{'c': 7, 'd': 8}
>>> vars(Z())
{'c': 7, 'd': 8}
```

Alternate type() Initialization

- Case Study: BltTypeEx.py

```
class zclass:
    def __init__(self, **kwargs):
        self.times = 1000
        print(f'Created zclass {kwargs}')

normal = zclass(times=3000)
print('1',vars(normal))

other = type('zclass', tuple(), dict(times=9000))
print('2',vars(other))
```

Python Meta

Try this @home?

```
>>> for a in copyright, credits, license:
        print(a)

>>> help("this")

>>> import antigravity
```

▶ False	▶ None	▶ True	▷ 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

KA1061: Integer Values

Beginner

```
>>> int(-255)
```

(1) Exception
(2) -False
(3) True
(4) 255
(5) -255

KA2036: Object Values

Intermediate

We use vars() to:

(1) Create object dictionaries
(2) Manage collection types
(3) Access 'dunder dict' values
(4) Manage string values
(5) Manage integral values

KA2037: Boolean Values

Intermediate

```
>>> bool(-255)

(1) Exception
(2) -False
(3) True
(4) 255
(5) -255
```

KA2038: List Comprehension

Intermediate

```
>>> [c for c in dir(7) if not c[0] == '_']
```

(1) Range Exception
(2) All public members
(3) []
(4) All private operations
(5) None of the above

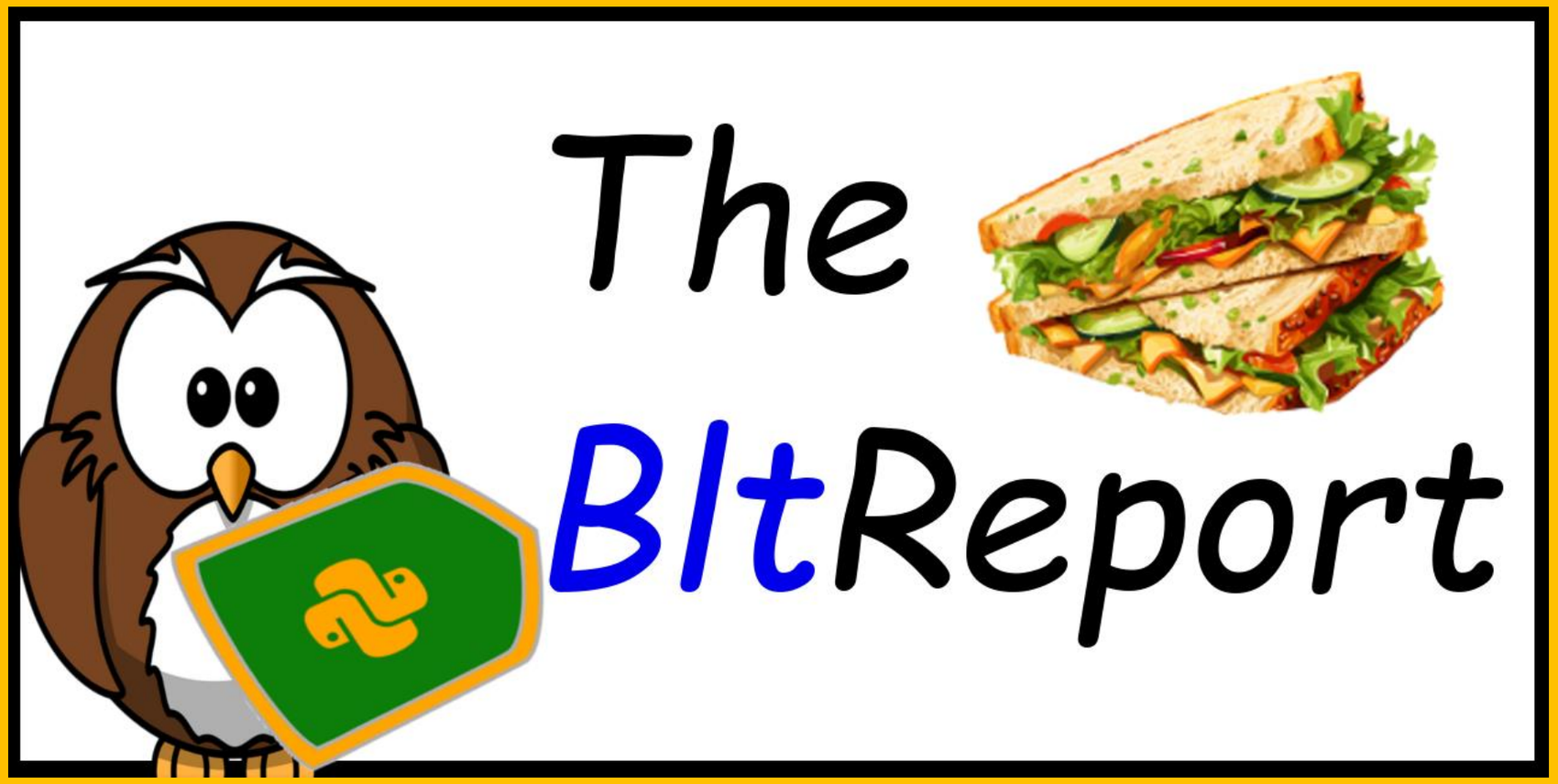
KA3036: Type Management

Advanced

Use type() to:

(1) Change existing members
(2) Create Objects
(3) Safely remove presence
(4) Determine instance type
(5) Two of the above

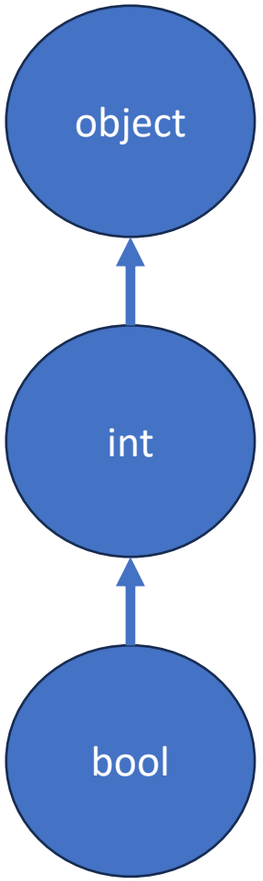
Video: BLT_00300



▶ False	▶ None	▶ True	▷ 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

‘isa’ == isinstance()

- Instance ~to~ Recipe(s)
  - isinstance(True, int)
  - isinstance(7, bool)



issubclass()

- Recipe ~to~ Recipe(s)
  - issubclass(bool, int)
  - issubclass(int, bool)

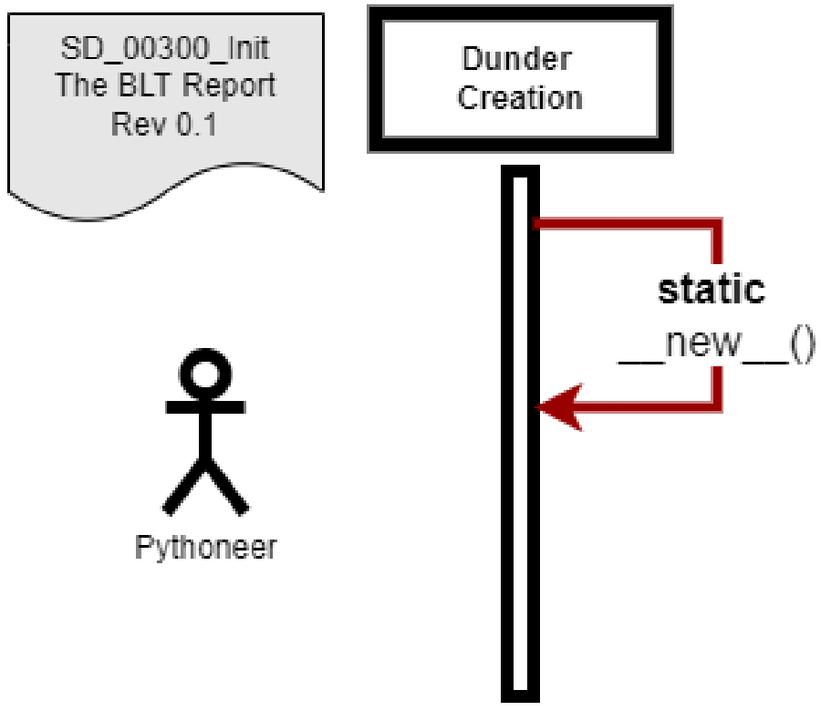
Review: type() Initialization

- Case Study: BltTypeEx.py

```
class zclass:
    def __init__(self, **kwargs):
        self.times = 1000
        print(f'Created zclass {kwargs}')

normal = zclass(times=3000)
print('1',vars(normal))

other = type('zclass', tuple(), dict(times=9000))
print('2',vars(other))
```

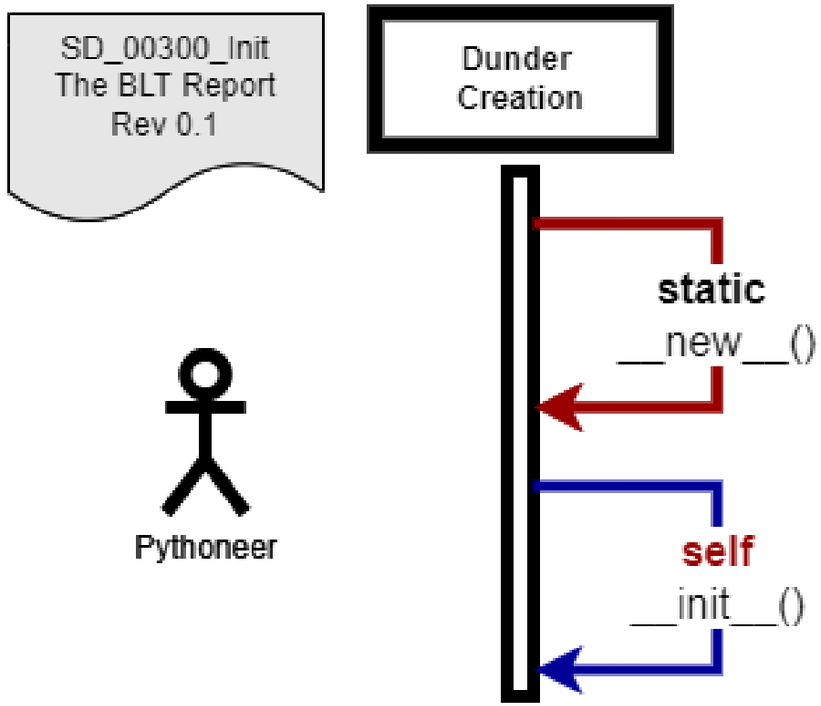
Review: Classic Initialization …

- Case Study: BltTypeEx.py



Review: Classic Initialization

- Case Study: BltTypeEx.py



The “Four ‘Atters”

- Safe – Always True / False
  - SET
  - HAS
- ‘Exceptional’
  - GET
    - Member **Requested** / AttributeError
  - DEL
    - **None** / AttributeError

Built-In setattr() ...

```
>>> help(setattr)
Help on built-in function setattr in module builtins:

setattr(obj, name, value, /)
    Sets the named attribute on the given object to the specified value.

    setattr(x, 'y', v) is equivalent to ``x.y = v''

>>> z = Z()
>>> setattr(z, 'a', 'Bingo!')
>>> z.a
'Bingo!'
```

Functional setattr()

```
>>> help(setattr)
Help on built-in function setattr in module builtins:

setattr(obj, name, value, /)
    Sets the named attribute on the given object to the specified value.

    setattr(x, 'y', v) is equivalent to ``x.y = v''

>>> class Z:
        pass

>>> z = Z()
>>> setattr(z, 'a', lambda: 'Howdie!')
>>> z.a()
'Howdie!'
```

Built-In hasattr()

```
>>> help(hasattr)
Help on built-in function hasattr in module builtins:

hasattr(obj, name, /)
    Return whether the object has an attribute with the given name.

    This is done by calling getattr(obj, name) and catching AttributeError.

>>> z = Z()
>>> setattr(z, 'a', lambda: 'Howdie!')
>>> hasattr(z, 'a')
True
>>> hasattr(z, 'b')
False
```

getattr()

- “GET” is Imperative?

```
>>> help(getattr)
Help on built-in function getattr in module builtins:

getattr(...)
    getattr(object, name[, default]) -> value

    Get a named attribute from an object; getattr(x, 'y') is equivalent to x.y.
    When a default argument is given, it is returned when the attribute doesn't
    exist; without it, an exception is raised in that case.

>>> setattr(z, 'a', lambda: 'Howdie!')
>>> getattr(z, 'a')
<function <lambda> at 0x000001EEB4E0CDC0>
>>> getattr(z, 'a')()
'Howdie!'
```

getattr()

- “GET” is Imperative
  - Exceptional!

```
>>> hasattr(7, 'a')
False
>>> getattr(7, 'a')
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    getattr(7, 'a')
AttributeError: 'int' object has no attribute 'a'
```

delattr()

- Attribute Removal

```
>>> help(delattr)
Help on built-in function delattr in module builtins:

delattr(obj, name, /)
    Deletes the named attribute from the given object.

    delattr(x, 'y') is equivalent to ``del x.y''

>>> class Z:
        pass

>>> z = Z()
>>> z.a = True
>>> delattr(z, 'a')
```

delattr()

- “DELETE” is Imperative?
  - Exceptional!

```
>>> delattr(7, 'a')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    delattr(7, 'a')
AttributeError: 'int' object has no attribute 'a'
```

Concept: ‘Weak References’?

- More: Python Docs

“A weak reference to an object is **not enough** to keep the object alive …

A primary use for weak references is to implement caches or mappings holding large objects, where it’s desired that a large **object not be kept alive** solely because it appears in a cache or mapping.”

- False
- all
- bool
- callable
- complex
- dict
- eval
- float
- globals
- hex
- isinstance
- license
- max
- object
- pow
- range
- set
- staticmethod
- tuple
- None
- any
- breakpoint
- chr
- copyright
- dir
- exec
- format
- hasattr
- id
- issubclass
- list
- memoryview
- oct
- print
- repr
- setattr
- str
- type
- True
- ascii
- bytearray
- classmethod
- credits
- divmod
- exit
- frozenset
- hash
- input
- iter
- locals
- min
- open
- property
- reversed
- slice
- sum
- vars
- abs
- bin
- bytes
- compile
- delattr
- enumerate
- filter
- getattr
- help
- int
- len
- map
- next
- ord
- quit
- round
- sorted
- super
- zip

KA2039: Instance Detection

Intermediate

```
>>> isinstance()
```

(1) Checks `isa` relationships
(2) Accepts exactly two parameters
(3) Always requires >= 1 class
(4) Returns True or False
(5) All of the above

KA2040: Class Detection

Intermediate

```
>>> issubclass()
```

(1) Checks `isa` relationships
(2) Accepts exactly two parameters
(3) Always requires >= 1 class
(4) Returns True or False
(5) All of the above

KA2041: Class Relations

Intermediate

```
>>> All Python classes have:

(1) No common ancestor
(2) A common `int` ancestor
(3) A common `str` ancestor
(4) Multiple inheritance
(5) None of the above
```

KA3031: Object Management

Advanced

How to check object membership?

(1) Use Inheritance
(2) Use 'Duck Typing'
(3) Use hasattr()
(4) Use getattr()
(5) Either 3 or 4

KA3032: Object Management

Advanced

How to add object membership?

(1) Use Inheritance
(2) Use 'Duck Typing'
(3) Use hasattr()
(4) Use setattr()
(5) Either 3 or 4

KA3033: Object Management

Advanced

Use hasattr() to:

(1) Create Inheritance
(2) Create 'Duck Typing'
(3) Safely check presence
(4) Return named attribute
(5) All of the above

KA3034: Object Management

Advanced

Use getattr() to:

(1) Check Inheritance
(2) Create 'Duck Typing'
(3) Safely check presence
(4) Return named attribute
(5) All of the above

KA3035: Object Management

Advanced

Use setattr() to:

(1) Replace existing member
(2) Assign new attribute
(3) Safely check presence
(4) Attempt attribute removal
(5) Two of the above

KA3036: Object Management

Advanced

Use delattr() to:

(1) Replace existing member
(2) Assign new attribute
(3) Safely remove presence
(4) Attempt attribute removal
(5) None of the above

KA3037: Type Management

Advanced

Use type() to:

(1) Change existing members
(2) Create Objects
(3) Safely remove presence
(4) Determine instance type
(5) Two of the above

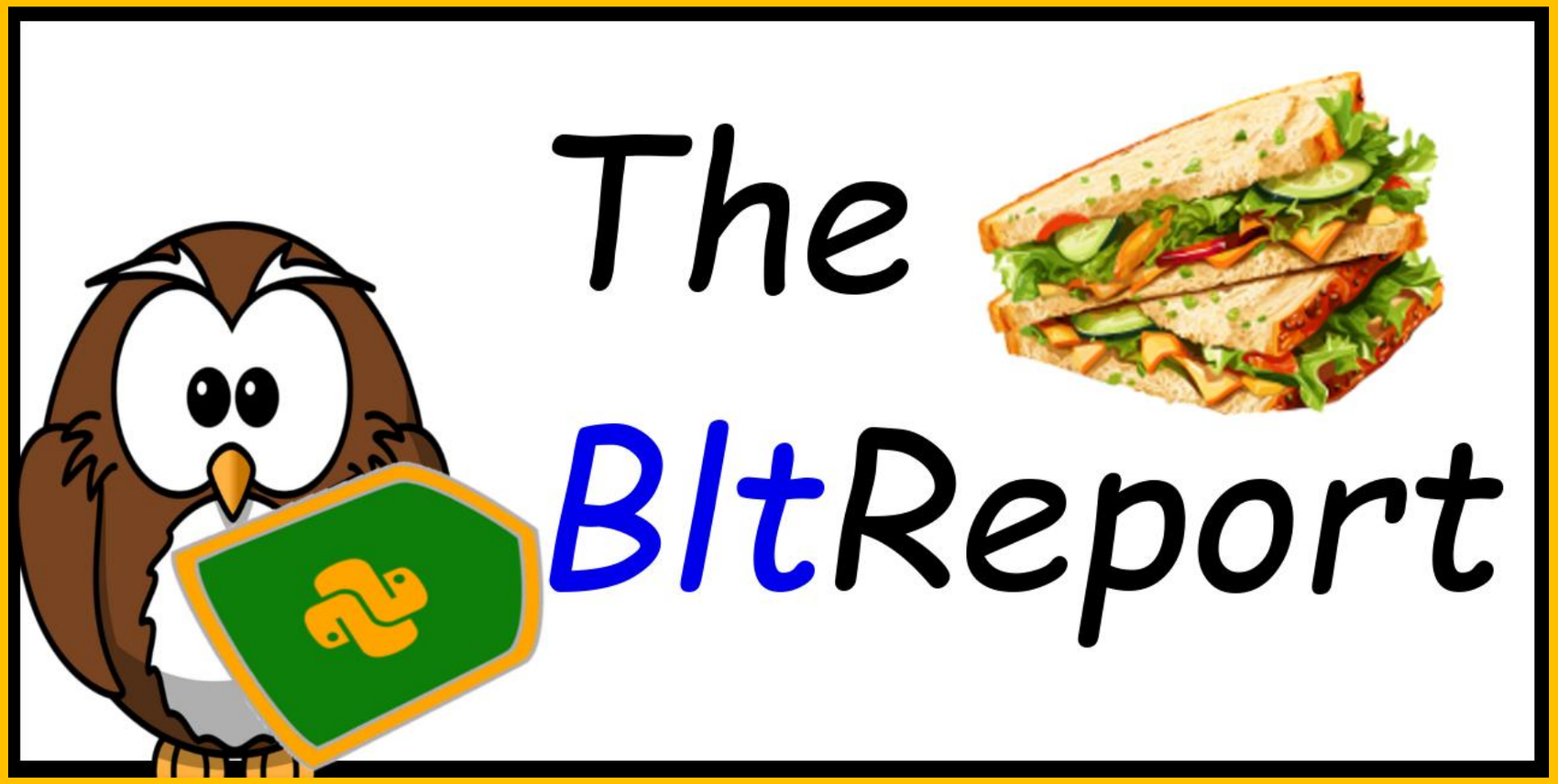
KA3038: Object Management

Advanced

A 'Weak Reference':

(1) Changes existing members
(2) Creates Objects
(3) Safely removes members
(4) Determines instance type
(5) None of the above

Video: BLT_00400



- False
- all
- bool
- callable
- complex
- dict
- eval
- float
- globals
- hex
- isinstance
- license
- max
- object
- pow
- range
- set
- staticmethod
- tuple
- None
- any
- breakpoint
- chr
- copyright
- dir
- exec
- format
- hasattr
- id
- issubclass
- list
- memoryview
- oct
- print
- repr
- setattr
- str
- type
- True
- ascii
- bytearray
- classmethod
- credits
- divmod
- exit
- frozenset
- hash
- input
- iter
- locals
- min
- open
- property
- reversed
- slice
- sum
- vars
- abs
- bin
- bytes
- compile
- delattr
- enumerate
- filter
- getattr
- help
- int
- len
- map
- next
- ord
- quit
- round
- sorted
- super
- zip

Classic enumerate()

```
class enumerate(object)
 |  enumerate(iterable, start=0)
 |
 |  Return an enumerate object.
 |
 |    iterable
 |      an object supporting iteration
 |
 |  The enumerate object yields pairs containing a count (from start, which
 |  defaults to zero) and a value yielded by the iterable argument.
```

```
>>> type(enumerate)
<class 'type'>
```

Iterable Types?

```
>>> set(dir(enumerate('Nagy'))) - \
    set(dir(object()))
{'__next__', '__iter__', '__class_getitem__'}
>>>
```

▶ False	▶ None	▶ True	▷ 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

KA1063: Enumerations

Beginner

Built-in enumerate()

(1) Is a class
(2) Is a function

KA1064: Iteration

Beginner

Which types are iterable?

(1) int()
(2) str()
(3) float()
(4) bool()
(5) None of the above

KA1065: Iteration

Beginner

Which types are iterable?

(1) dict()
(2) str()
(3) list()
(4) set()
(5) All of the above

KA3039: Iteration Management

Advanced

Iterable classes:

(1) Cannot be created
(2) Are part of object()
(3) Are part of all built-ins
(4) Determines instance type
(5) None of the above

KA3040: Iteration Management

Advanced

Iterables must implement __next__

(1) True
(2) False

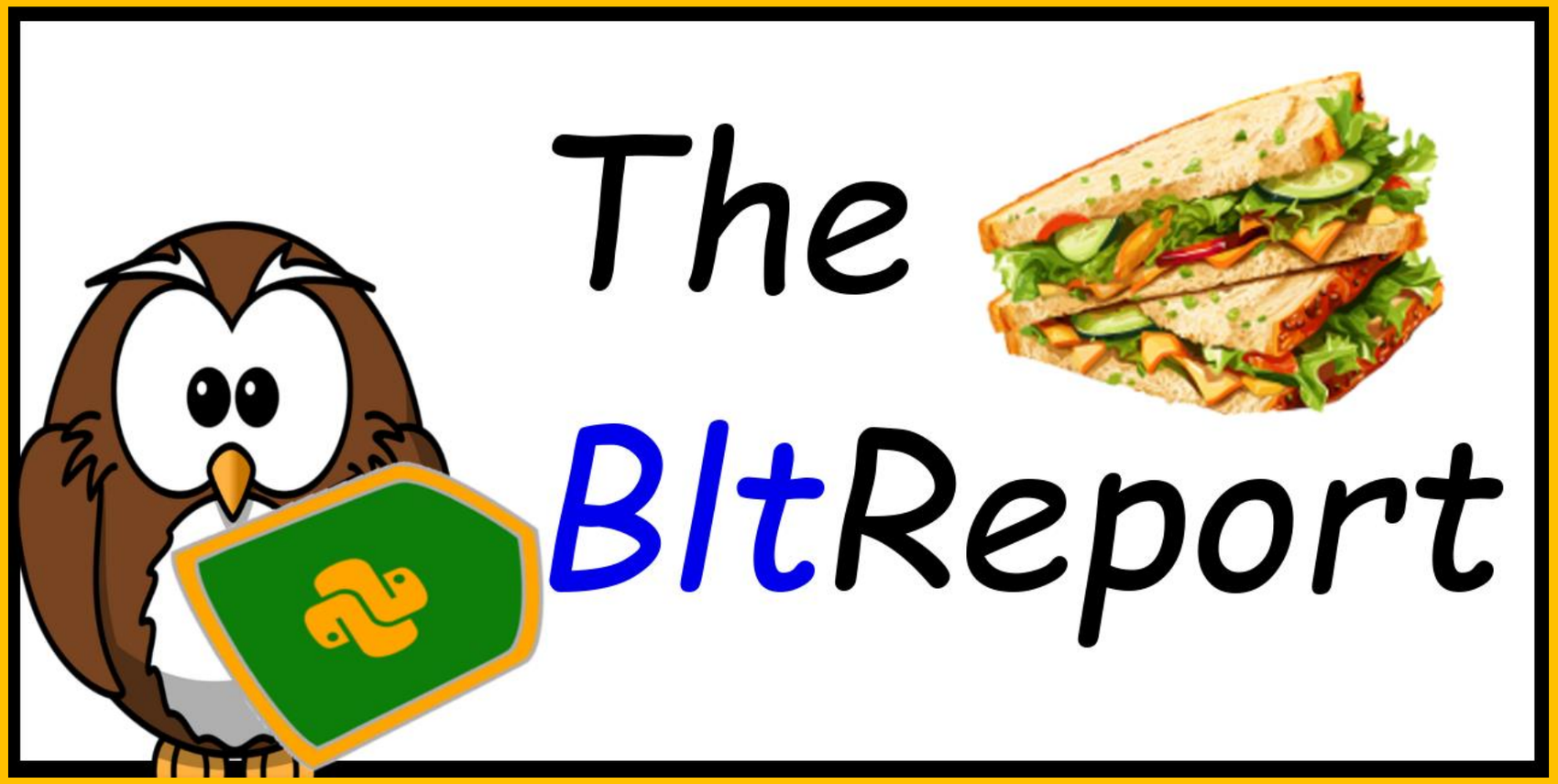
KA3041: Iteration Management

Advanced

Iterables must implement __iter__

(1) True
(2) False

Video: BLT_00500



Focus: range(…)

- False
- all
- bool
- callable
- complex
- dict
- eval
- float
- globals
- hex
- isinstance
- license
- max
- object
- pow
- range
- set
- staticmethod
- tuple
- None
- any
- breakpoint
- chr
- copyright
- dir
- exec
- format
- hasattr
- id
- issubclass
- list
- memoryview
- oct
- print
- repr
- setattr
- str
- type
- True
- ascii
- bytearray
- classmethod
- credits
- divmod
- exit
- frozenset
- hash
- input
- iter
- locals
- min
- open
- property
- reversed
- slice
- sum
- vars
- abs
- bin
- bytes
- compile
- delattr
- enumerate
- filter
- getattr
- help
- int
- len
- map
- next
- ord
- quit
- round
- sorted
- super
- zip

Demo: Using range(…)

- Three Forms
  - Zero-Based (default)
  - Offset Specified
  - Stepped Values

Ranged Instance Representation

```
>>> type(range(5))
<class 'range'>

>>> range(5)
range(0, 5)
```

Iterable Types?

```
>>> set(dir(enumerate('Nagy'))) - \
    set(dir(object()))
{'__next__', '__iter__', '__class_getitem__'}
>>>
```

Ranged Report …

```
z = set(dir(range(5))) - set(dir(object()))
z = sorted(z)
for ss, item in enumerate(z):
    if ss % 2 == 0:
        print()
    print(f'{ss+1:>02} {item:<16}', end='')
```

Ranged Delta

```
01 __bool__
02 __contains__
03 __getitem__
04 __iter__
05 __len__
06 __reversed__
07 count
08 index
09 start
10 step
11 stop
```

Commons: Range & Enumerate

```
r = set(dir(range(5)))
e = set(dir(enumerate('')))
z = sorted(r.intersection(e))
for ss, item in enumerate(z):
    if ss % 2 == 0:
        print()
    print(f'{ss+1:>02} {item:<20}', end='')
```

```
15 __lt__                  16 __ne__
17 __new__                 18 __reduce__
19 __reduce_ex__           20 __repr__
21 __setattr__             22 __sizeof__
23 __str__                 24 __subclasshook__
```

KA1028: Slicing Range

Beginner

How to enumerate the odd numbers from:

```
t = range(10, 16)
```

(1) `print(enumerate(t, 1, 2))`
(2) `print(*t[1::2])`
(3) `print(range(t, 1, 2))`
(4) `print(t[1::2])`
(5) None of the above

KA1066: Range Rep

Beginner

```
>>> range(5)
```

(1) range(5)
(2) range(1,6)
(3) range(1,5)
(4) range(0,5)
(5) None of the above

KA1067: Modulus -v- Range

Beginner

Show every odd between 1 and 10

```
(1) print(*range(1,11,2))
(2) for i in range(11):
        if i % 3 == 0:
            print(i)
(3) for i in range(1,11,1):
        if i % 3 == 0:
            print(i)
(4) range(1,10,3)
(5) Two of the above
```

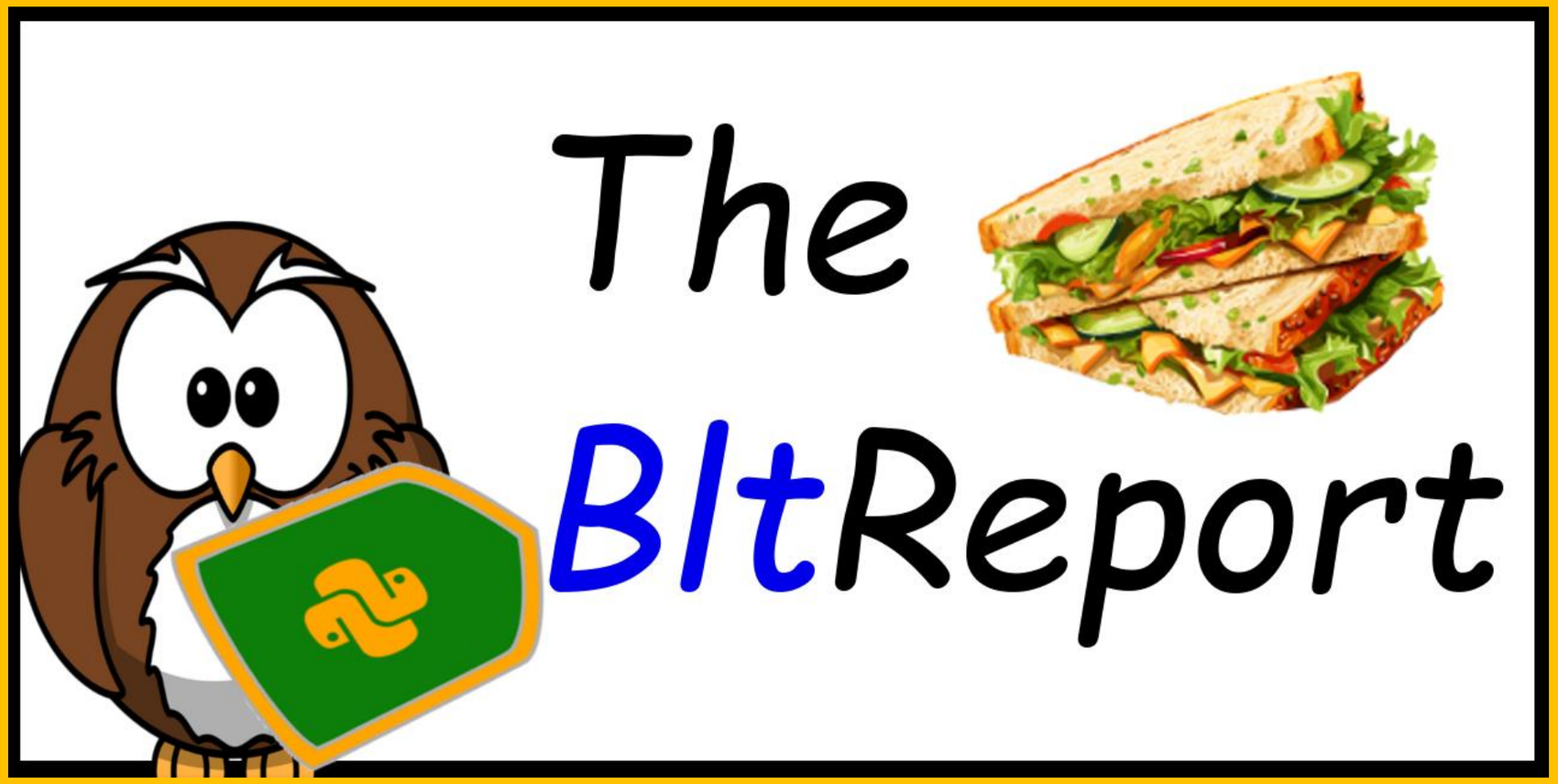
KA2030: Range -v- Enumerate

Intermediate

What is a difference between range() and enumerate()?

(1) range() returns 2 values
(2) enumerate() returns 3 values
(3) enumerate() has three forms / types
(4) range() offers a step / increment value
(5) None of the above

Video: BLT_00600



- False
- all
- bool
- callable
- complex
- dict
- eval
- float
- globals
- hex
- isinstance
- license
- max
- object
- pow
- range
- set
- staticmethod
- tuple
- None
- any
- breakpoint
- chr
- copyright
- dir
- exec
- format
- hasattr
- id
- issubclass
- list
- memoryview
- oct
- print
- repr
- setattr
- str
- type
- True
- ascii
- bytearray
- classmethod
- credits
- divmod
- exit
- frozenset
- hash
- input
- iter
- locals
- min
- open
- property
- reversed
- slice
- sum
- vars
- abs
- bin
- bytes
- compile
- delattr
- enumerate
- filter
- getattr
- help
- int
- len
- map
- next
- ord
- quit
- round
- sorted
- super
- zip

Demo: repr()

```
>>> help(repr)
Help on built-in function repr in module builtins:

repr(obj, /)
    Return the canonical string representation of the object.

    For many object types, including most builtins, eval(repr(obj)) == obj.
```

Demo: id()

```
>>> help(id)
Help on built-in function id in module builtins:

id(obj, /)
    Return the identity of an object.

    This is guaranteed to be unique among simultaneously existing objects.
    (CPython uses the object's memory address.)
```

Demo: str()

```
class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
```

String Caching?

```
>>> s = 'Nagy'
>>> id(s)
2051767269744
>>> x = 'Nagy'
>>> id(x)
2051767269744
```

```
>>> del s;del x
>>> s = 'Nagy'
>>> id(s)
2051769189936
```

String Encoding

```
class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
```

```
>>> import sys
>>> sys.getdefaultencoding()
'utf-8'
```

KA1009: Object Identification

Beginner

The built-in id() function will:

(1) Compare object identities
(2) Verify object content
(3) Uniquely identify an object
(4) Identify the Python version

KA2014: Class Representation

Intermediate

The purpose of the repr() built-in function is to ?

(1) Reproduce two objects by combination
(2) Return a cloned-instance of an object
(3) Return a string for humans to view
(4) Get a string to re-create an object
(5) None of the above

KA2042: Object Evaluation

Intermediate

The difference between str() and repr():

(1) str() objects use eval()
(2) eval() objects use str()
(3) repr() is for humans
(4) Strings are always the same
(5) None of the above

KA2043: Object Contents

Intermediate

A __str__ opportunity may represent:

(1) Instance reporting
(2) Object evaluation
(3) Big Data
(4) SQL Databases
(5) None of the above

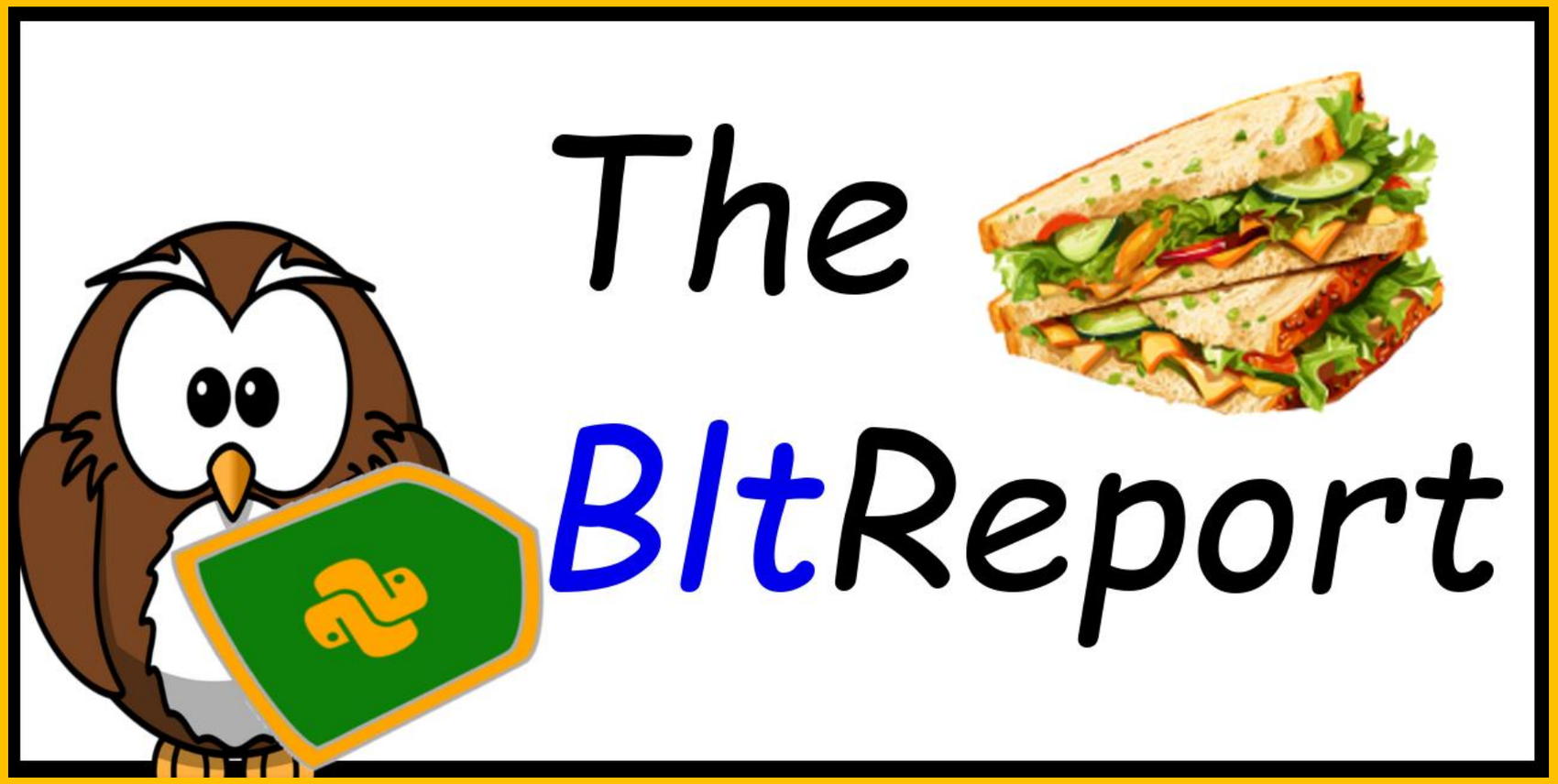
KA3043: String Encoding

Advanced

How to determine platform str() encoding?

(1) import os;sys.getdefaultencoding()
(2) import sys;os.getdefaultencoding()
(3) import sys;sys.getdefaultencoding()
(4) import str;sys.getdefaultencoding()
(5) import str;str.getdefaultencoding()

Video: BLT_00700



- False
- None
- True
- 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

Exec ~v~ Eval: No help()?

```
>>> help(eval)
Help on built-in function eval in module builtins:

eval(source, globals=None, locals=None, /)
    Evaluate the given source in the context of globals and locals.

    The source may be a string representing a Python expression
    or a code object as returned by compile().
    The globals must be a dictionary and locals can be any mapping,
    defaulting to the current globals and locals.
    If only globals is given, locals defaults to it.
```

-> object

```
>>> help(exec)
Help on built-in function exec in module builtins:

exec(source, globals=None, locals=None, /)
    Execute the given source in the context of globals and locals.

    The source may be a string representing one or more Python statements
    or a code object as returned by compile().
    The globals must be a dictionary and locals can be any mapping,
    defaulting to the current globals and locals.
    If only globals is given, locals defaults to it.
```

-> None

Locals? Globals?

- locals()
- globals()

```
>>> type(locals())
<class 'dict'>
>>> type(globals())
<class 'dict'>
```

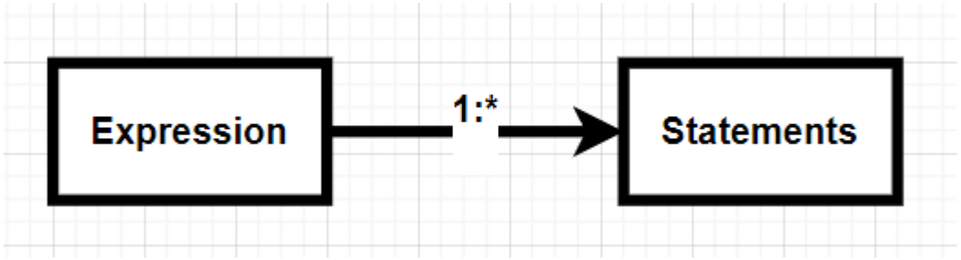
locals() –v globals()

```
>>> help(globals)
Help on built-in function globals in module builtins:

globals()
    Return the dictionary containing the current scope's global variables.

    NOTE: Updates to this dictionary *will* affect name lookups in the current
    global scope and vice-versa.

>>> help(locals)
Help on built-in function locals in module builtins:

locals()
    Return a dictionary containing the current scope's local variables.

    NOTE: Whether or not updates to this dictionary will affect name lookups in
    the local scope and vice-versa is *implementation dependent* and not
    covered by any backwards compatibility guarantees.
```

-> dict()

Expressions ~v~ Statements

- An Expression may have many Statements, but a Statement cannot.
  - Python “Expr Nodes”



Eval Environment Preserved

```
>>> g={'a':4};c = eval('{"a":a+7}', g)
>>> c
{'a': 11}
>>> g['a']
4
>>>
```

Exec Environment Updated

```
>>> g={'a':0};c = exec('a+=1', g)
>>> c
>>> print(c)
None
>>> g
{'a': 1, '__builtins__': {'__name__'
exceptions, and other objects.\n\nNo
```

Bottom-Line Concepts

- Use
  - Expressions

    exec if you do not need the return value

    Environment automatically updates
  - Statements

    eval if return is desired

    Environment not automatically updated

```
>>> print(eval('[1,2,3]'))
[1, 2, 3]
>>> print(exec('a=[1,2,3]'))
None
```

Expression exec()

- Eval: ‘Expression Exception’

```
>>> exec('a=[1,2,3]')
>>> print(eval('a=[1,2,3]'))
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    print(eval('a=[1,2,3]'))
  File "<string>", line 1
    a=[1,2,3]
      ^
SyntaxError: invalid syntax
```

KA2045: Code Evauation

Intermediate

The built-in eval():

(1) Can override globals()
(2) Can use custom locals()
(3) Always returns a result
(4) Can be used by compile()
(5) All of the above

KA3042: Code Evaluators

Advanced

Difference between eval() and exec()?

(1) eval() parses strings
(2) exec() runs programs
(3) Both are used by compile()
(4) eval() returns a result
(5) exec() returns a result

KA3045: Globals

Advanced

Pythons built-in global() returns a

(1) string
(2) boolean
(3) mutable copy
(4) mutable reference
(5) potential object

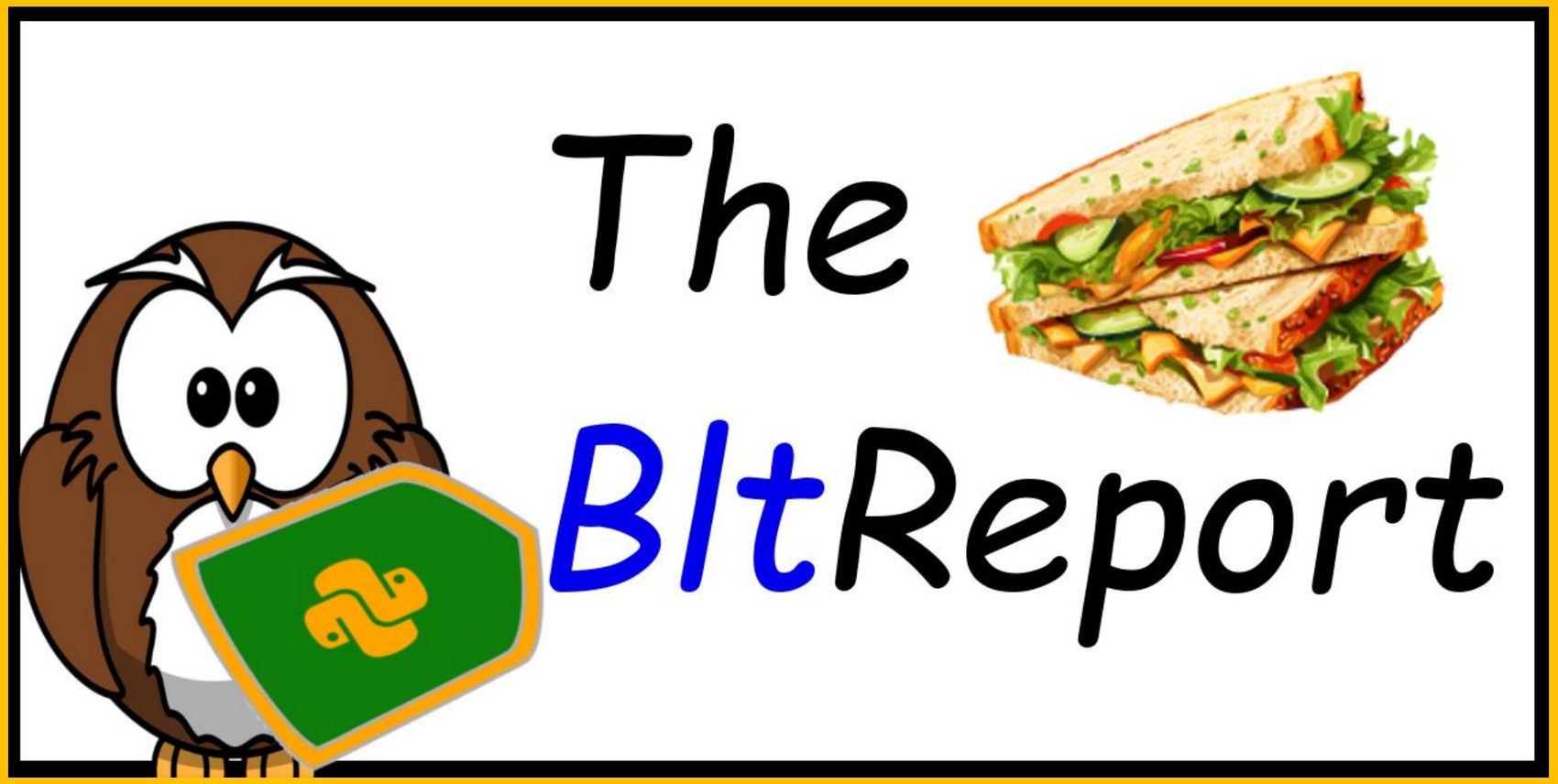
KA3046: Expressions -v- Statements

Advanced

An Expression may have many Statements, but a Statement cannot have many Expressions.

(1) True
(2) False

Video: BLT_00800



▶ False	▶ None	▶ True	▷ 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

Info: compile()

```
>>> help(compile)
Help on built-in function compile in module builtins:

compile(source, filename, mode, flags=0, dont_inherit=False, optimize=-1, *, _fea
ture_version=-1)
    Compile source into a code object that can be executed by exec() or eval().

    The source code may represent a Python module, statement or expression.
    The filename will be used for run-time error messages.
    The mode must be 'exec' to compile a module, 'single' to compile a
    single (interactive) statement, or 'eval' to compile an expression.
    The flags argument, if present, controls which future statements influence
    the compilation of the code.
    The dont_inherit argument, if true, stops the compilation inheriting
    the effects of any future statements in effect in the code calling
    compile; if absent or false these statements do influence the compilation,
    in addition to any features explicitly specified.
```

Return Type

```
>>> for mode in 'single', 'eval', 'exec':
        print(type(compile('[123]','',mode)))


<class 'code'>
<class 'code'>
<class 'code'>
```

Realizing compile()d Results

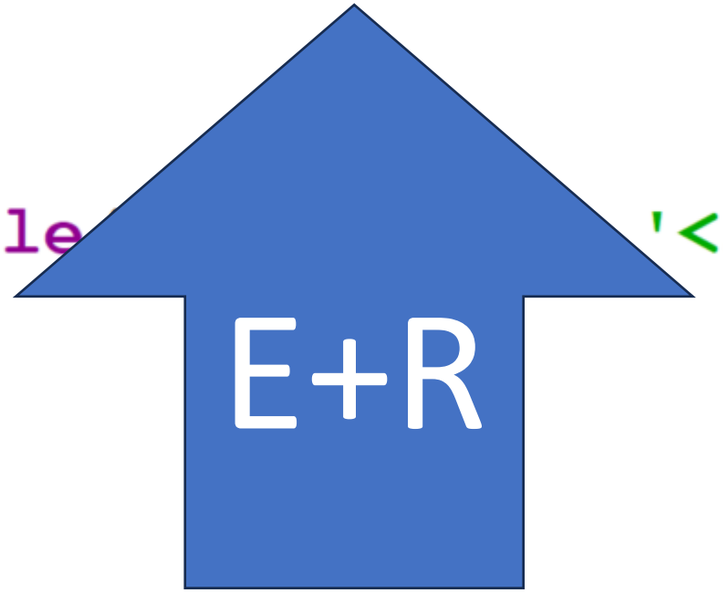
```
>>> compile('[1,2,3]','','exec')
<code object <module> at 0x000001D429AA2240, file "", line 1>

>>> compile('[1,2,3]','','eval')
<code object <module> at 0x000001D429AA22F0, file "", line 1>

>>> c = compile('[1,2,3]','','exec')
>>> print(eval(c))
None

>>> c = compile('[1,2,3]','','eval')
>>> print(eval(c))
[1, 2, 3]
```

Statement: ‘eval’

```
>>> a=4;eval(compile('a+5','<string>','eval'))
9
>>> a
4

>>> a=4;eval(compile('a*=10;a+5','<string>','eval'))
Traceback (most recent call last):
  File "<pyshell#142>", line 1, in <module>
    a=4;eval(compile('a*=10;a+5','<string>','eval'))
  File "<string>", line 1
    a*=10;a+5
     ^
SyntaxError: invalid syntax
```

Expression: ‘single’ and ‘exec’

```
>>> a=4;eval(compile('a*=10;a+5','<string>','single'))
45
>>> a
40
>>> a=4;eval(compile
                                    '<string>','exec'))
>>> a
40
```

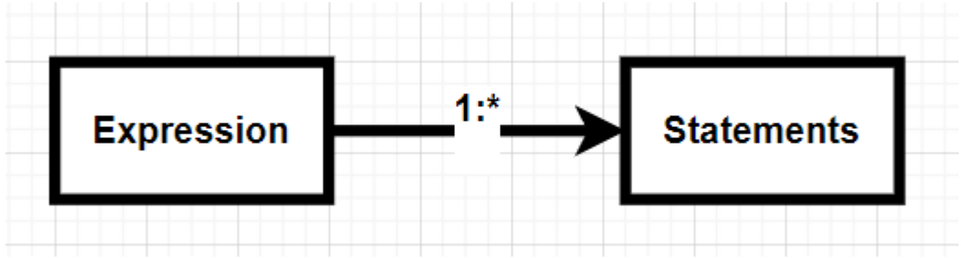


Module Importation

```
>>> eval(compile('import MightyMaxims','<user>','single'))
>>> eval(compile('import MightyMaxims','<user>','exec'))
>>> eval(compile('import MightyMaxims','<user>','eval'))
Traceback (most recent call last):
  File "<pyshell#114>", line 1, in <module>
    eval(compile('import MightyMaxims','<user>','eval'))
  File "<user>", line 1
    import MightyMaxims
    ^
SyntaxError: invalid syntax
```

Which compile()?

- Importing Modules
  - ‘**single**’
    - Compile an Expression
    - Compile a Module
    - Effects & Results
  - ‘**eval**’
    - Compile a Statement (“what if”)
    - Results
  - ‘**exec**’
    - Compile an Expression (“make it so”)
    - Compile a Module
    - Effects



Parameter Opportunities

- ✓Source
  - Unicode String
  - Byte String
  - Abstract Syntax Tree (AST) Object
    - Note: ast.parse also converts code into AST
- Filename
- Mode
- Flags
- Optimize

Parameter Opportunities

- Source

✓Filename

  - File from whence above came
  - ELSE some distinct tag / name
  - ‘<string>’ (common)

- Mode
- Flags
- Optimize

Parameter Opportunities

- Source
- Filename
- ✓Mode
  - ‘single’, ‘eval’, ‘exec’
- Flags
- Optimize

Parameter Opportunities

- Source
- Filename
- Mode
- ✓Flags (optional arguments)
  - **don’t_inherit** - future features
  - **flags** - Compiler options
- Optimize

Parameter Opportunities

- ✓Optimize – Integral Values
  - CLI Default (-1)
  - None (0)
    - __debug__ is True
  - One (1)
    - __debug__ is False
    - Remove assert()ions
  - Two (2)
    - Also removes docstrings

KA3044: Compile

Advanced

Pythons built-in compile()

(1) May use 'eval()' or 'exec()'
(2) Parameters allow locals()
(3) Parameters allow globals()
(4) Requires a file-name
(5) Compiles & executes code

KA3046: Expressions -v- Statements

Advanced

An Expression may have many Statements, but a Statement cannot have many Expressions.

(1) True
(2) False

KA3047: Expressions -v- Statements

Advanced

An Expression may have many Statements, but a Statement cannot have many Statements

(1) True
(2) False

KA3048: Compile

Advanced

Pythons built-in compile()

(1) 'eval', 'exec', 'single'
(2) 'debug', 'eval', 'single'
(3) 'compile', 'run', 'debug'
(4) 'secure', 'lock', 'unlock'
(5) Compiles & executes code

KA3049: Compile

Advanced

Pythons built-in compile()

(1) Returns a 'code class'
(2) Realized using exec()
(3) Realized using eval()
(4) Two of the above
(5) All of the above

KA3050: Compile

Advanced

```
a=4;eval(compile('a+=5','','exec'))
```

(1) a == 9
(2) a == 4
(3) SyntaxError
(4) Two of the above
(5) None of the above

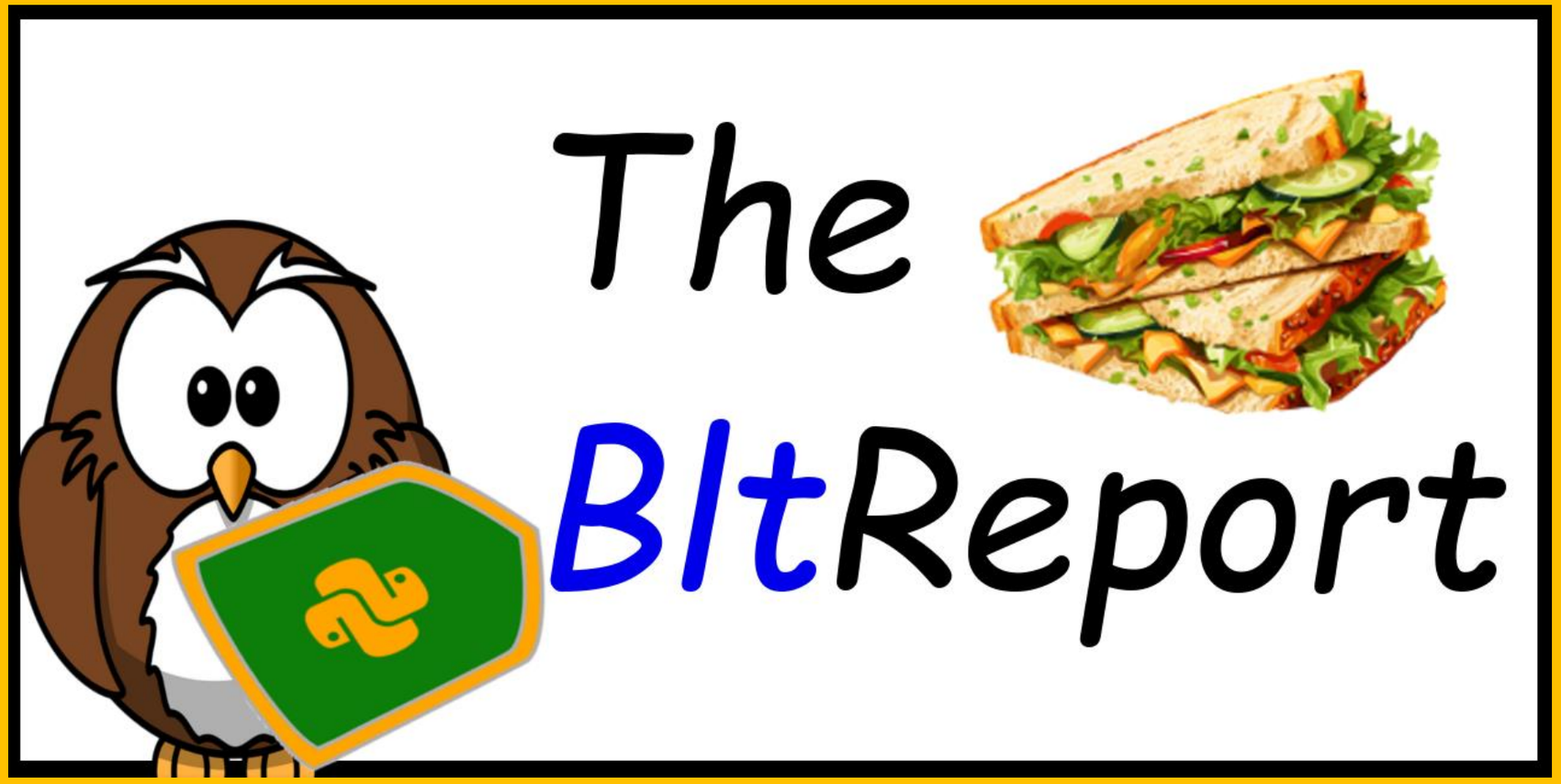
KA3051: Compile

Advanced

```
a=4;eval(compile('a+=5','','eval'))
```

(1) a == 9
(2) a == 4
(3) SyntaxError
(4) Two of the above
(5) None of the above

Video: BLT_00900



▶ False	▶ None	▶ True	▷ 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

```
>>> help(staticmethod)
Help on class staticmethod in module builtins:

class staticmethod(object)
 |  staticmethod(function) -> method
 |
 |  Convert a function to be a static method.
 |
 |  A static method does not receive an implicit first argument.
 |  To declare a static method, use this idiom:
 |
 |       class C:
 |           @staticmethod
 |           def f(arg1, arg2, ...):
 |               ...
 |
 |  It can be called either on the class (e.g. C.f()) or on an instance
 |  (e.g. C().f()). Both the class and the instance are ignored, and
 |  neither is passed implicitly as the first argument to the method.
```

Review: D.I.Y Decorators …

```
class Z:
    def __init__(self, func):
        print(type(func), func.__name__)
        self.a_func = func

    def __call__(self):
        print('Calling', self.a_func.__name__)
        results = self.a_func()
        print('Returning', results)
        return results
```

File: DiyDecorator.py

Static @Z

```
>>> class B:
        @Z
        def foo(): return 5


<class 'function'> foo
>>> B.foo()
Calling foo
Returning 5
5
```

Review: Decorated ‘Ops

- ‘Ops In-Action:

```
>>> type(Z)
<class 'type'>
>>> type(staticmethod)
<class 'type'>
```

BONUS: D.I.Y Decorated ‘Ops

- ‘Ops In-Action:

```
>>> @Z
def foo(): ...

<class 'function'> foo
>>> isinstance(foo, Z)
True
>>> callable(Z)
True
>>> callable(foo)
True
>>> foo()
Calling foo
Returning None
>>>
```

@staticmethod ~v~ D.I.Y

```
>>> @staticmethod
def foo(): ...

>>> callable(foo)
False
```

Using @staticmethod …

```
class S:
    def __init__(self): ...

    @staticmethod
    def foo():
        print('Spam! =)')
```

OPTIONAL

CALLABLE

File: StaticDecorator.py

Using @staticmethod

```
>>> S()
<__main__.S object at 0x0000018C8DF75A60>
>>> S()()
Returning 5
5
>>> S.foo()
Spam! =)
>>> s = S()
>>> s.foo()
Spam! =)
```

Where's the … Function?

```
>>> s = staticmethod(lambda : 'Zoom')
>>> s.__func__()
'Zoom'
>>>
>>> @staticmethod
def foo(): return 'zoom!'

>>> foo.__func__()
'zoom!'
```

```
>>> type(s)
<class 'staticmethod'>
```

KA3003: exec()

Advanced

What is Python's built-in exec()function used for?

(1) To run an external application
(2) To check if an object is callable
(3) To evaluate and test Python code
(4) To change environment variables
(5) None of the above

KA3008: Function Call Management

Advanced

Which Meta Method(s) are required to create a callable object?

KA3009: Function Call Detection

Advanced

Which built-in function allows us to determine if an instance is callable?

KA3019: Function Decoration

Advanced

What is a `decorator function`?

KA3023: Static Methods

Advanced

What is the purpose of a @staticmethod?

KA3052: @staticmethod

Advanced

The staticmethod decorator can be used on non-class functions

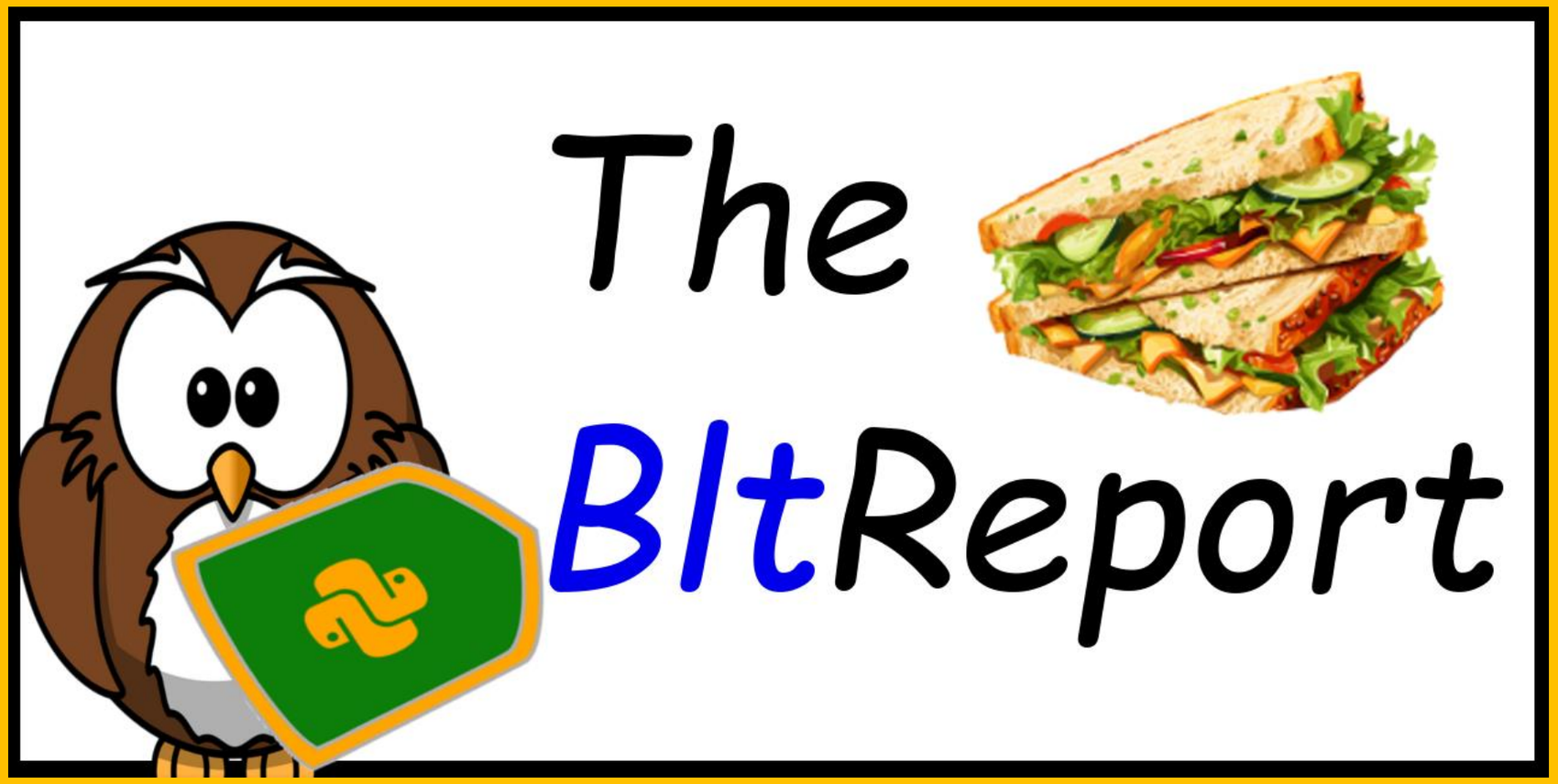
(1) True
(2) False

KA3053: Decorators

Advanced

How would you create a decorator?

Video: BLT_01000



- False
- all
- bool
- callable
- complex
- dict
- eval
- float
- globals
- hex
- isinstance
- license
- max
- object
- pow
- range
- set
- staticmethod
- tuple
- None
- any
- breakpoint
- chr
- copyright
- dir
- exec
- format
- hasattr
- id
- issubclass
- list
- memoryview
- oct
- print
- repr
- setattr
- str
- type
- True
- ascii
- bytearray
- classmethod
- credits
- divmod
- exit
- frozenset
- hash
- input
- iter
- locals
- min
- open
- property
- reversed
- slice
- sum
- vars
- abs
- bin
- bytes
- compile
- delattr
- enumerate
- filter
- getattr
- help
- int
- len
- map
- next
- ord
- quit
- round
- sorted
- super
- zip

```
class classmethod(object)
 |  classmethod(function) -> method
 |
 |  Convert a function to be a class method.
 |
 |  A class method receives the class as implicit first argument,
 |  just like an instance method receives the instance.
 |  To declare a class method, use this idiom:
 |
 |    class C:
 |        @classmethod
 |        def f(cls, arg1, arg2, ...):
 |            ...
 |
 |  It can be called either on the class (e.g. C.f()) or on an instance
 |  (e.g. C().f()).  The instance is ignored except for its class.
 |  If a class method is called for a derived class, the derived class
 |  object is passed as the implied first argument.
 |
 |  Class methods are different than C++ or Java static methods.
 |  If you want those, see the staticmethod builtin.
```

```
>>> help(staticmethod)
Help on class staticmethod in module builtins:

class staticmethod(object)
 |  staticmethod(function) -> method
 |
 |  Convert a function to be a static method.
 |
 |  A static method does not receive an implicit first argument.
 |  To declare a static method, use this idiom:
 |
 |       class C:
 |           @staticmethod
 |           def f(arg1, arg2, ...):
 |               ...
 |
 |  It can be called either on the class (e.g. C.f()) or on an instance
 |  (e.g. C().f()). Both the class and the instance are ignored, and
 |  neither is passed implicitly as the first argument to the method.
```

Review: __func__

```
>>> s = staticmethod(lambda : 'Zoom')
>>> s.__func__()
'Zoom'
>>>
>>> @staticmethod
def foo(): return 'zoom!'

>>> foo.__func__()
'zoom!'
```

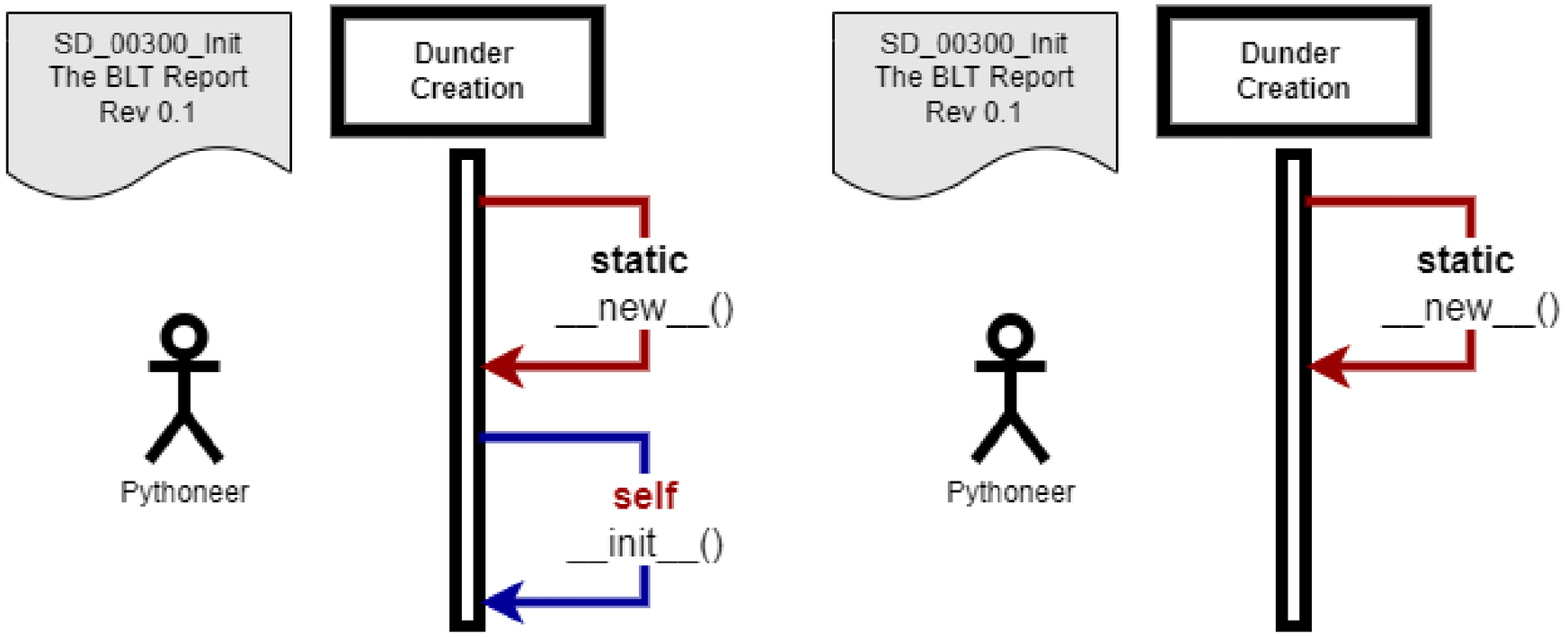
Same Expectation (“Framework”)

```
>>> c = classmethod(lambda : 'Same!')
>>> c.__func__()
'Same!'
>>>
```

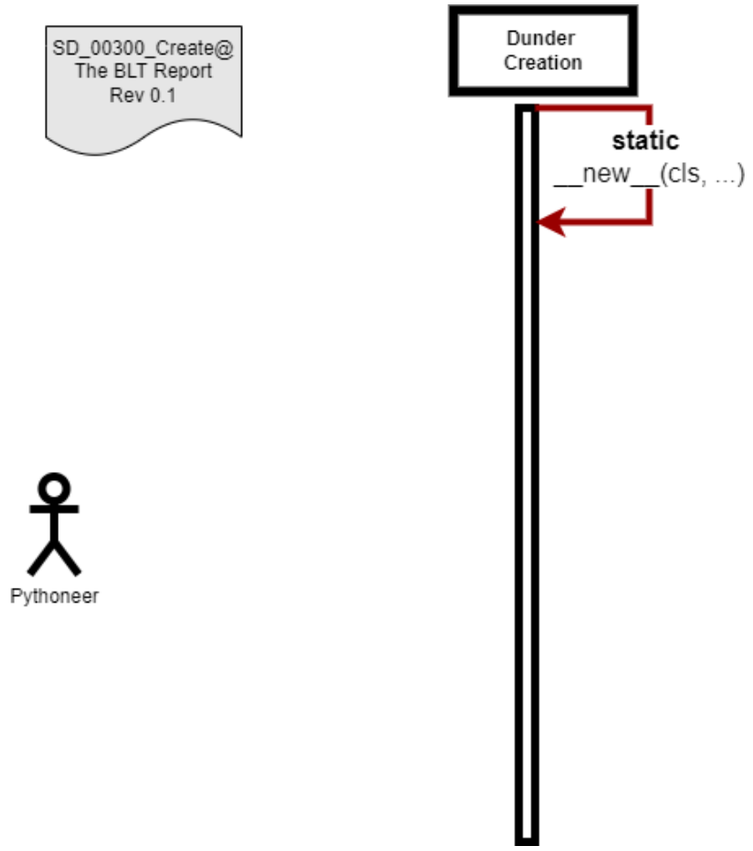
```
class S:
    def __init__(self, msg='Default'):
        if msg: print(msg)

    @classmethod
    def foo(cls):
        return cls('classM')

    @staticmethod
    def __call__():
        return S('staticM')
```

File: ClassDecorator.py

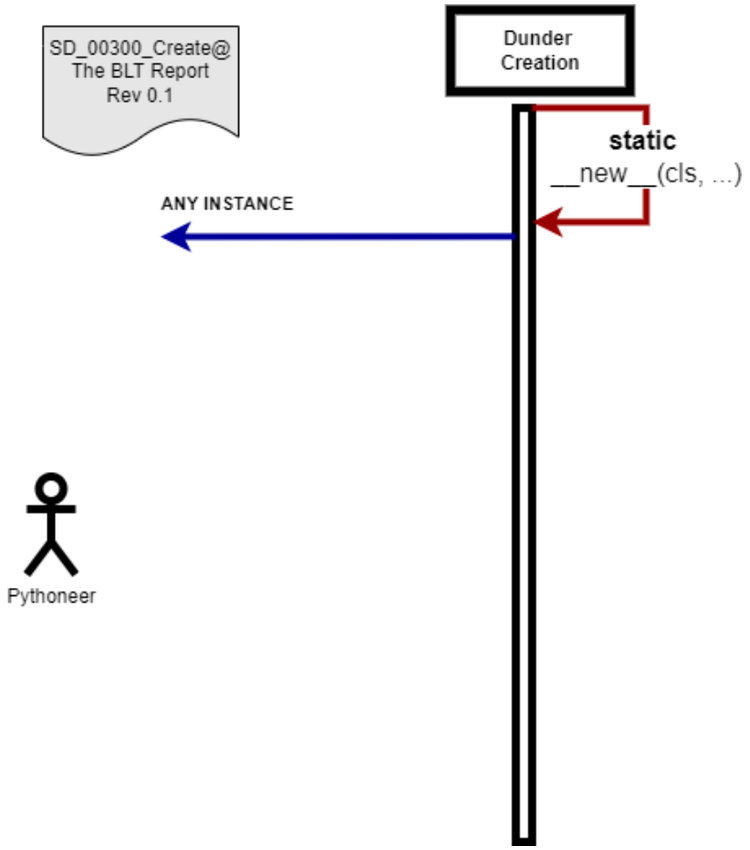
Review: Classic Initialization



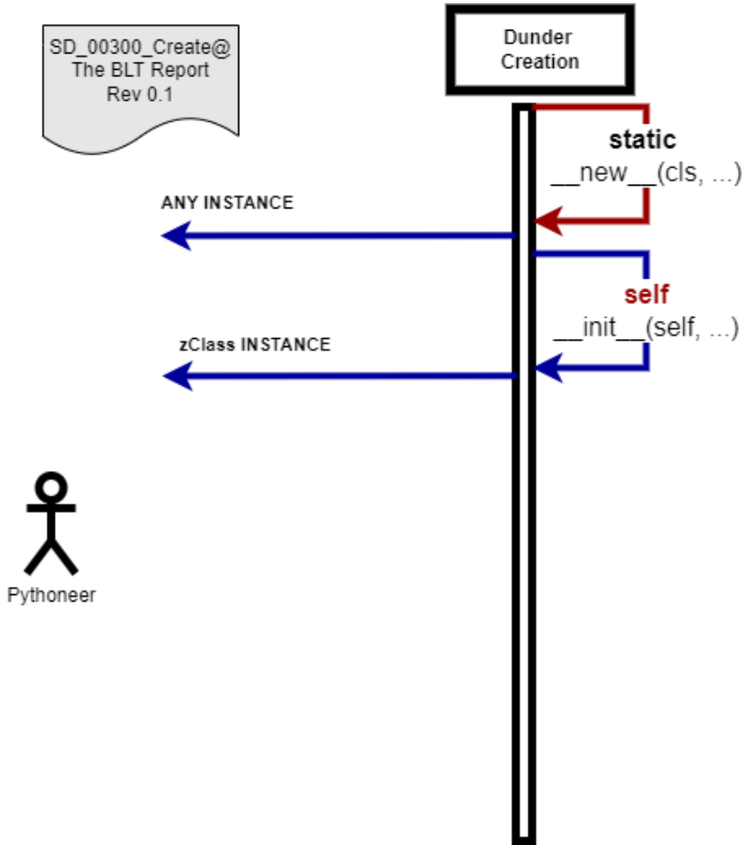
__new__(cls, ...)


SD_00300_Create@
The BLT Report
Rev 0.1
Dunder
Creation
static
__new__(cls, ...)
Pythoneer

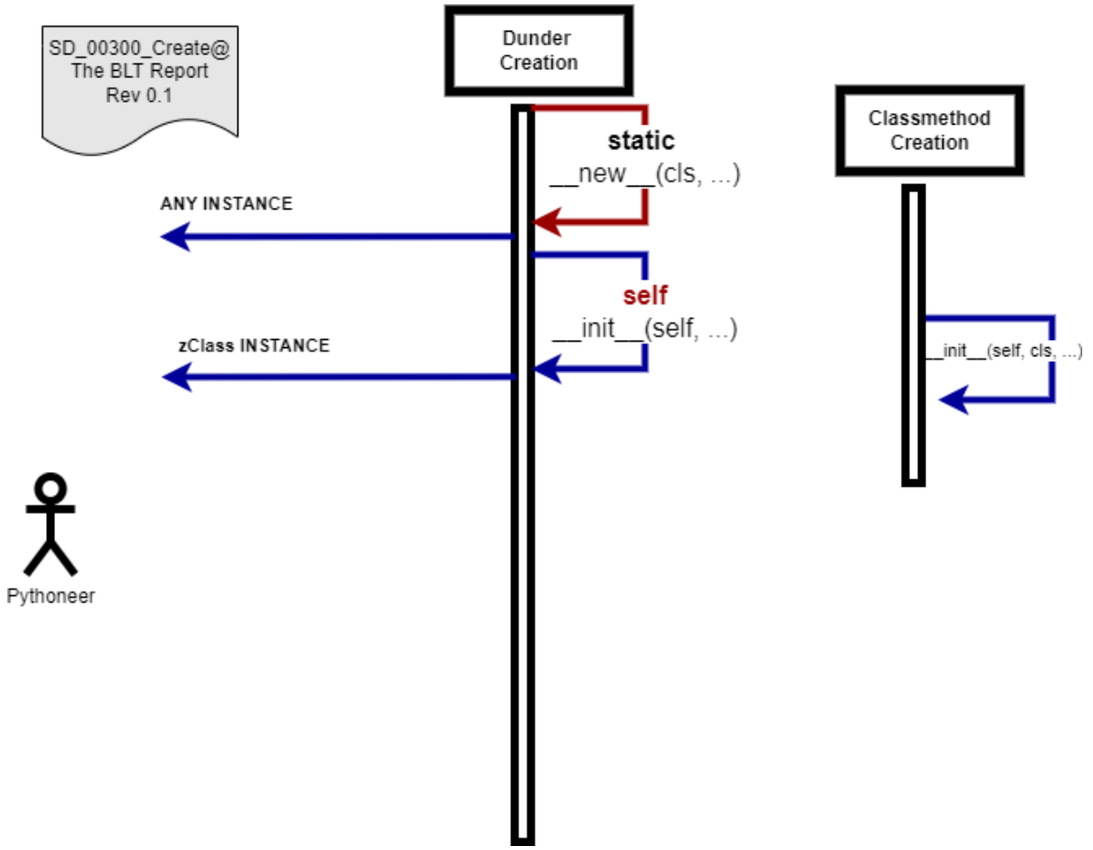
Possible … !


SD_00300_Create@
The BLT Report
Rev 0.1
Dunder
Creation
static
__new__(cls, ...)
ANY INSTANCE
Pythoneer

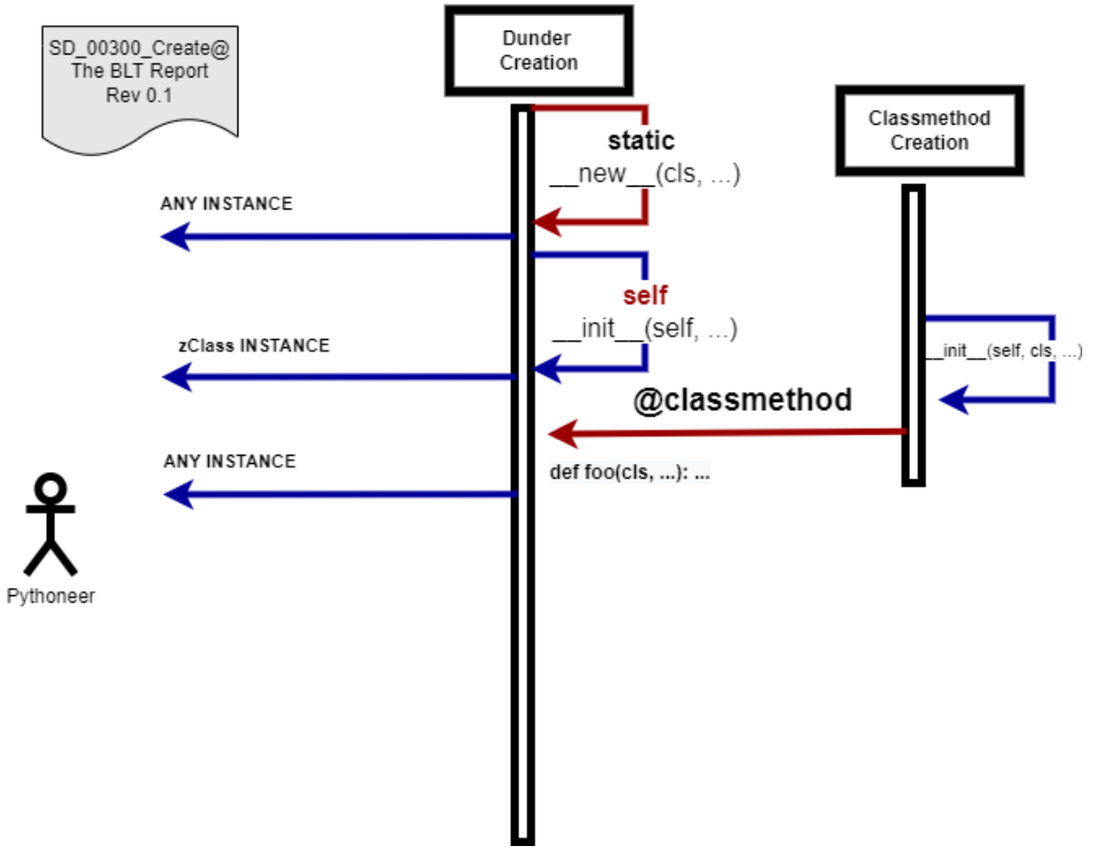
Usually, Same Type ...



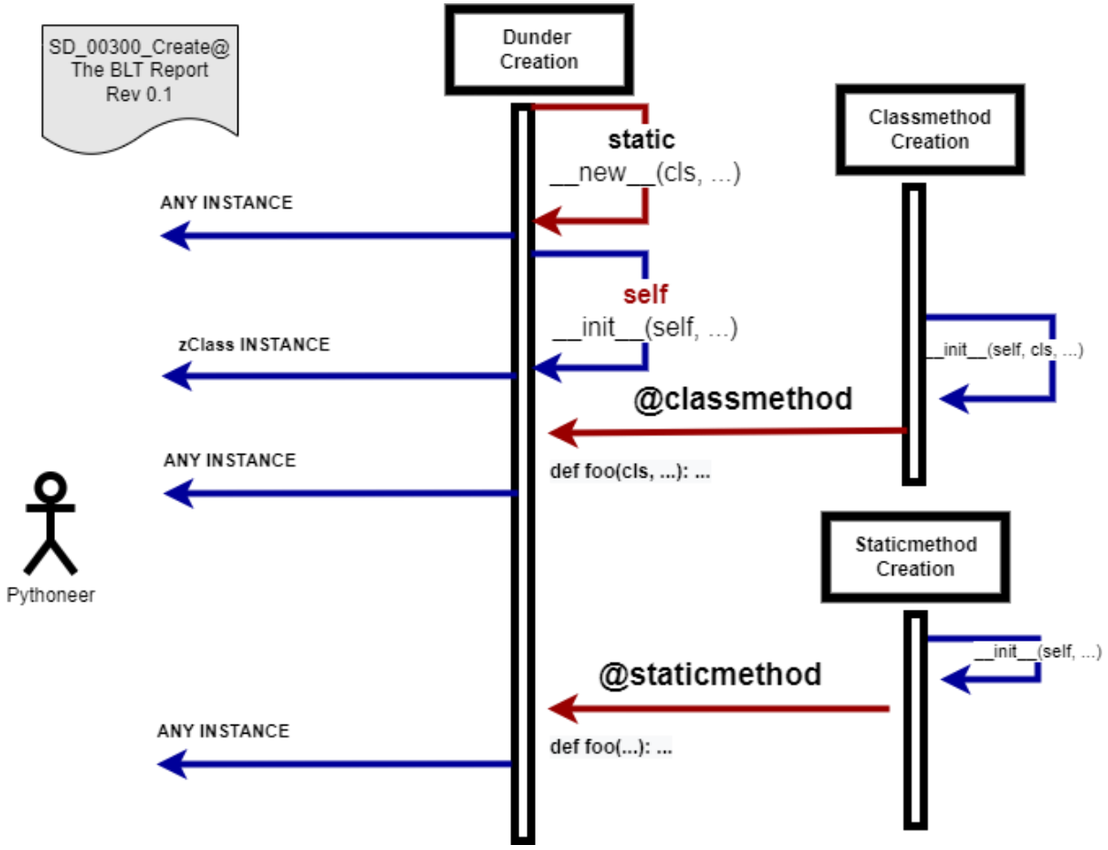
Same cls Type ...



Anything ~via~ __func__ ‘Ops



Anything ~via~ __func__ ‘Ops



Creation Opportunities

- @staticmethod
  - Might return ***anything***
  - More parameters may be provided
  - No parameters required

- @classmethod
  - Requisite parameter is ‘type’
  - More parameters may be provided
  - Also could return ***anything***!

KA3021: Static Construction

Advanced

What is the purpose of a @classmethod?

KA3022: @classmethod -v- new

Advanced

What parameters do a @classmethod and the new dunder have in common?

KA3025: Object Factories

Advanced

What is the purpose of an `Object Factory`?

KA2005: Basic Inheritance

Intermediate

In Python 'inheritance' is a way to ?

(1) Manage object factories
(2) Manage meta-data
(3) Assign interfaces
(4) Ensure meta relationships
(5) None of the above

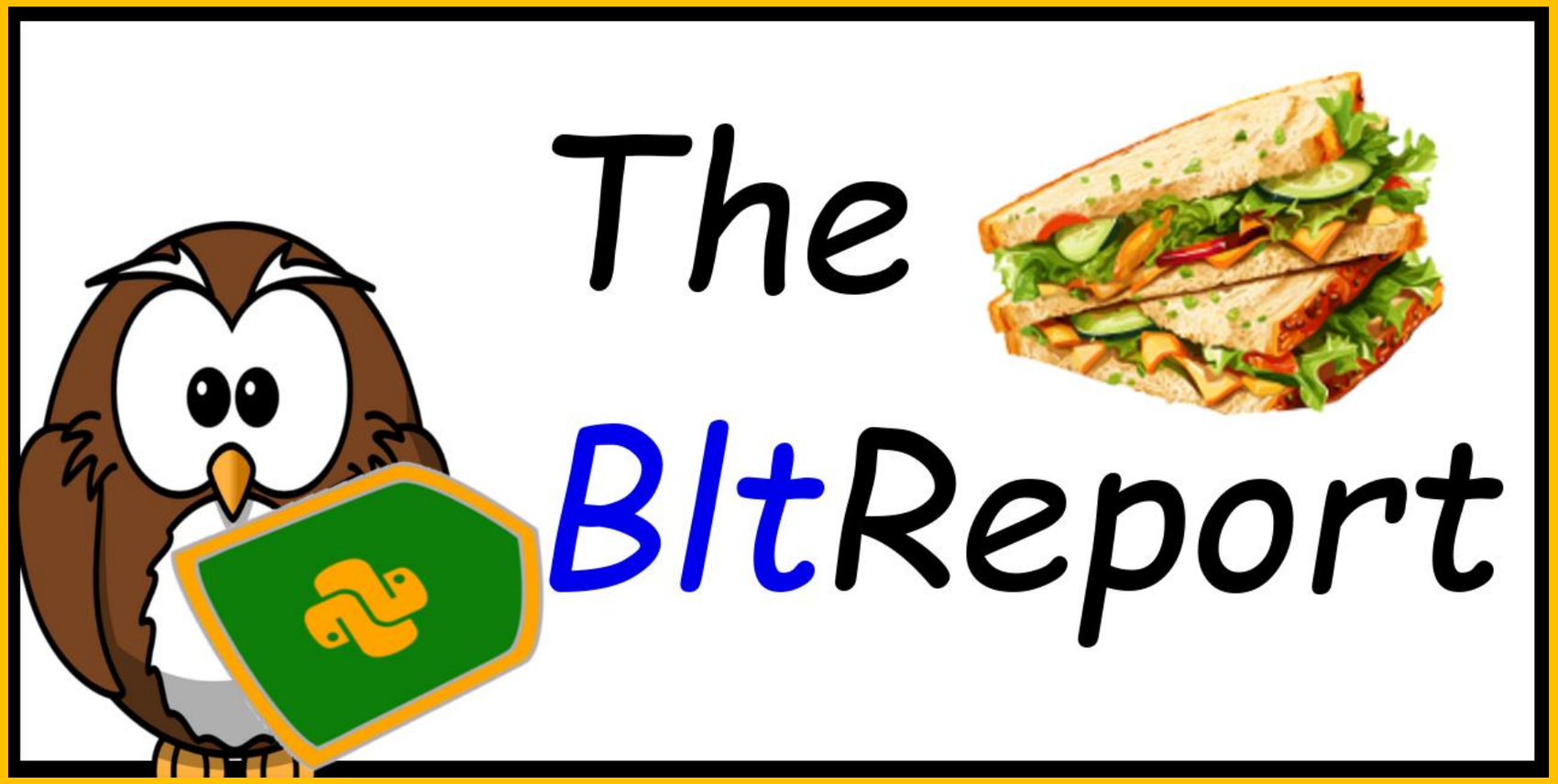
KA2015: Framework Basics

Intermediate

The purpose of a 'framework' is to ?

(1) Enforce the C.R.U.D design pattern
(2) Provide a testable infrastructure
(3) Provide common code re-use patterns
(4) Ensure object-factor relationships
(5) None of the above

Video: BLT_01100



▶ False	▶ None	▶ True	▷ 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

Rpt: Public ‘Ops

```
>>> Rpt.type_report(Rpt)
01 common_report 02 get_private    03 get_public     04 get_set
05 reporter      06 resolve        07 type_report
```

File: TypeReport.py

/The-Built-In-Report/study/TypeReport.py

```
>>> Rpt.type_report(set)
01 add                                 02 clear
03 copy                                04 difference
05 difference_update                   06 discard
07 intersection                        08 intersection_update
09 isdisjoint                          10 issubset
11 issuperset                          12 pop
13 remove                              14 symmetric_difference
15 symmetric_difference_update         16 union
17 update
```

Collection Commons

```
>>> _ = Rpt.common_report(dict, set)
01 clear   02 copy    03 pop     04 update

>>> _ = Rpt.common_report(list, set)
01 clear   02 copy    03 pop     04 remove
```

Review: Set Creations

- Element Ordering == Eccentric!

```
>>> set('Booya!')
{'!', 'a', 'y', 'o', 'B'}
>>> set(_)
{'y', 'o', '!', 'B', 'a'}
```

```
>>> set(sorted('Booya!'))
{'o', 'y', 'a', '!', 'B'}
```

Mathematicals

```
>>> a = set('cat')
>>> b = set('cat')
>>> id(a);id(b)
1598952245728
1598952244832
>>> a - b
set()
```

Set Operators

```
&,  &=
|,  |=
^,  ^=
-,  -=
>,  >=
<,  <=
```

01 add

```
>>> help(set.add)->None
Help on method_descriptor:

add(...)
    Add an element to a set.

    This has no effect if the element is already present.
```

02 clear

```
>>> help(set.clear)->None
Help on method_descriptor:

clear(...)
    Remove all elements from this set.
```

03 copy

```
>>> help(set.copy) ->set
Help on method_descriptor:

copy(...)
    Return a shallow copy of a set.
```

04 difference

```
>>> help(set.difference) ->set
Help on method_descriptor:

difference(...)
    Return the difference of two or more sets as a new set.

    (i.e. all elements that are in this set but not the others.)
```

set - other

05 difference_update

```
>>> help(set.difference_update)
Help on method_descriptor:

difference_update(...)
    Remove all elements of another set from this set.
```

set -= other

06 discard

```
>>> help(set.discard)->None
Help on method_descriptor:

discard(...)
    Remove an element from a set if it is a member.

    If the element is not a member, do nothing.
```

07 intersection

```
>>> help(set.intersection) ->set
Help on method_descriptor:

intersection(...)
    Return the intersection of two sets as a new set.

    (i.e. all elements that are in both sets.)
```

set & other

08 intersection_update

```
>>> help(set.intersection_update)
Help on method_descriptor:

intersection_update(...)
    Update a set with the intersection of itself and another.
```

set &= other

Caveat: Updates …

```
class Z:
  def __and__(self, param):
    print('gotcha.')

>>> Z() & Z()
gotcha.
```

Caveat, Updated!

```
class Z:
  def __and__(self, param):
      print('gotcha.')

>>> Z() &= Z()
```

SyntaxError: '**function call**' is an illegal expression for **augmented assignment**

```
>>> a = Z()
>>> b = Z()
>>> a &= b
gotcha.
```

```
>>> print(set('rat') & set('cat'))
{'t', 'a'}
>>> print(set('rat') &= set('cat'))
SyntaxError: invalid syntax
```

09 isdisjoint

```
>>> help(set.isdisjoint) ->bool
Help on method_descriptor:

isdisjoint(...)
    Return True if two sets have a null intersection.
```

10 issubset

```
>>> help(set.issubset) ->bool
Help on method_descriptor:

issubset(...)
    Report whether another set contains this set.
```

set <[=] other

11 issuperset

```
>>> help(set.issuperset)->bool
Help on method_descriptor:

issuperset(...)
    Report whether this set contains another set.
```

set >[=] other

12 pop

```
>>> help(set.pop)->object
Help on method_descriptor:

pop(...)
    Remove and return an arbitrary set element.
    Raises KeyError if the set is empty.
```

13 remove

```
>>> help(set.remove)->None
Help on method_descriptor:

remove(...)
    Remove an element from a set; it must be a member.

    If the element is not a member, raise a KeyError.
```

Item Removal

- .remove(object) – Raises Exceptions
- .discard(object) – “Just Do It”

✓Both return ‘**None**’

14 symmetric_difference ...

```
>>> help(set.symmetric_difference) ->set
Help on method_descriptor:

symmetric_difference(...)
    Return the symmetric difference of two sets as a new set.

    (i.e. all elements that are in exactly one of the sets.)
```

set ^ other

14 symmetric_difference

```
>>> set('yaboo!').difference(set('booya'))
{'!'}

>>> set('booya').difference(set('yaboo!'))
set()

>>> set('booya').symmetric_difference(set('yaboo!'))
{'!'}
```

15 symmetric_difference_update ...

```
>>> help(set.symmetric_difference_update) ->None
Help on method_descriptor:

symmetric_difference_update(...)
    Update a set with the symmetric difference of itself and another.
```

set ^= other

15 symmetric_difference_update

```
>>> set('yaboo!').difference(set('booya'))
{'!'}

>>> set('booya').difference(set('yaboo!'))
set()

>>> set('yaboo!').symmetric_difference_update(set('booya'))
>>> _
{'!'}
>>> set('booya').symmetric_difference_update(set('yaboo!'))
>>> _
{'!'}
```

16 union ...

```
>>> help(set.union) ->set
Help on method_descriptor:

union(...)
    Return the union of sets as a new set.

    (i.e. all elements that are in either set.)
```

set | other

16 union

```
>>> a = set('')
>>> b = a.union(set('Nagy'),set('egy'), set('booya!'))
>>> id(a);id(b)
1655906044160
1655906045504


>>> a
set()
>>> b
{'o', '!', 'N', 'g', 'y', 'b', 'e', 'a'}
```

17 update ...

```
>>> help(set.update)->None
Help on method_descriptor:

update(...)
    Update a set with the union of itself and others.
```

set |= other

17 update

```
>>> set('').union(set('Nagy'),set('egy'), set('booya!'))
{'o', '!', 'N', 'g', 'y', 'b', 'e', 'a'}
>>>
>>> set('').update(set('Nagy'),set('egy'), set('booya!'))
>>> _
{'o', '!', 'N', 'g', 'y', 'b', 'e', 'a'}
```

KA1032: Set Creation

Beginner

What is the difference between set('cat') and {'cat'}?

(1) Nothing.
(2) The number of elements will differ
(3) Both perform Venn operations
(4) The instance will be the same
(5) Their content will be the same

KA1033: Set Differences

Beginner

What is the Venn difference between set('cat') and set('rat')?

(1) Nothing.
(2) The number of elements will differ
(3) Either 'r' or 'c'
(4) Either 'cr' or 'rc'
(5) Their difference will be the same

KA1068: Set Basics

Beginner

```
>>> set('cat') - set('rat')

(1) {'a', 't'}
(2) {'t'}
(3) {'c'}
(4) {'c','t'}
(5) Two of the above
```

KA1069: Set Basics

Beginner

```
>>> set('rat') - set('cat')

(1) {'a', 't'}
(2) {'r'}
(3) {'c'}
(4) {'c','t'}
(5) Two of the above
```

KA2034: Set Comprehension

Intermediate

```
>>> {a for a in 'baggy' if a == 'g'}

(1) set()
(2) {'g', 'g'}
(3) {'g'}
(4) ['g']
(5) ['g', 'g']
```

KA3054: Advanced Set

Advanced

```
>>> set('cat') ^ set('rat')

(1) {'c'}
(2) {'t'}
(3) {'c', 't'}
(4) {'a'}
(5) {'a', 't'}
```

KA3055: Advanced Dunders

Advanced

How to intercept the `&` Operator?

(1) _&_
(2) __&__
(3) __amp__
(4) _amp_
(5) __and__

Video: BLT_01200

The

BltReport

False	None	True	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

Immutable … Frozen-Set?

```
>>> Rpt.type_report(set)
01 add                                  02 clear
03 copy                                 04 difference
05 difference_update                    06 discard
07 intersection                         08 intersection_update
09 isdisjoint                           10 issubset
11 issuperset                           12 pop
13 remove                               14 symmetric_difference
15 symmetric_difference_update          16 union
17 update
```

```
>>> issubclass(frozenset, set)
False
>>> issubclass(set, frozenset)
False
```

“Duck Typing”

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.

--- James Whitcomb Riley

Doctor Quote #73603

Classic Constructions

```
>>> frozenset('Whoopieee!')
frozenset({'!', 'e', 'o', 'i', 'W', 'h', 'p'})
>>>

>>> set(_)
{'o', 'i', '!', 'W', 'e', 'h', 'p'}
```

“Frozen” Set Operators

intersection	&,	&=
union	|,	|=
sym diff	^,	^=
difference	-,	-=

Creationals (NO UPDATES)

>,	>=
<,	<=

True/False

Keys – Hashable?

```
>>> frozenset(range(1,6))
frozenset({1, 2, 3, 4, 5})
>>> set([1,2,3])
{1, 2, 3}

>>> {[1,2,3]}
Traceback (most recent call last):
  File "<pyshell#72>", line 1, in <module>
    {[1,2,3]}
TypeError: unhashable type: 'list'
```

Getting hash()

```
>>> help(hash)
Help on built-in function hash in module builtins:

hash(obj, /)
    Return the hash value for the given object.

    Two objects that compare equal must also have the same hash value, but the
    reverse is not necessarily true.
```

Spelunking hash()

```
>>> Rpt.type_report(object())
False

>>> hash(object())
106799348275

>>> issubclass(list, object)
True

>>> hash(list())
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    hash(list())
TypeError: unhashable type: 'list'
```

KA2046: Frozenset

Intermediate

The frozenset():

(1) Is a built-in function
(2) Can be modified
(3) Is an instance of 'set'
(4) Is a subclass of 'set'
(5) None of the above

KA2047: Using Hash

Intermediate

The hash():

(1) Always returns a number
(2) Can be modified
(3) Is not required by set()
(4) Never raises an Exception
(5) None of the above

KA3026: Duck Typing

Advanced

What is `Duck Typing`?

KA3027: Duck Typing Caveats

Advanced

What are the pros & cons of `Duck Typing`?

KA3057: Hashing

Advanced

What is hashing?

(1) Creating a string for a number
(2) Creating a list for a string
(3) Creating a number for an object
(4) Raising a 'value error'
(5) All of the above

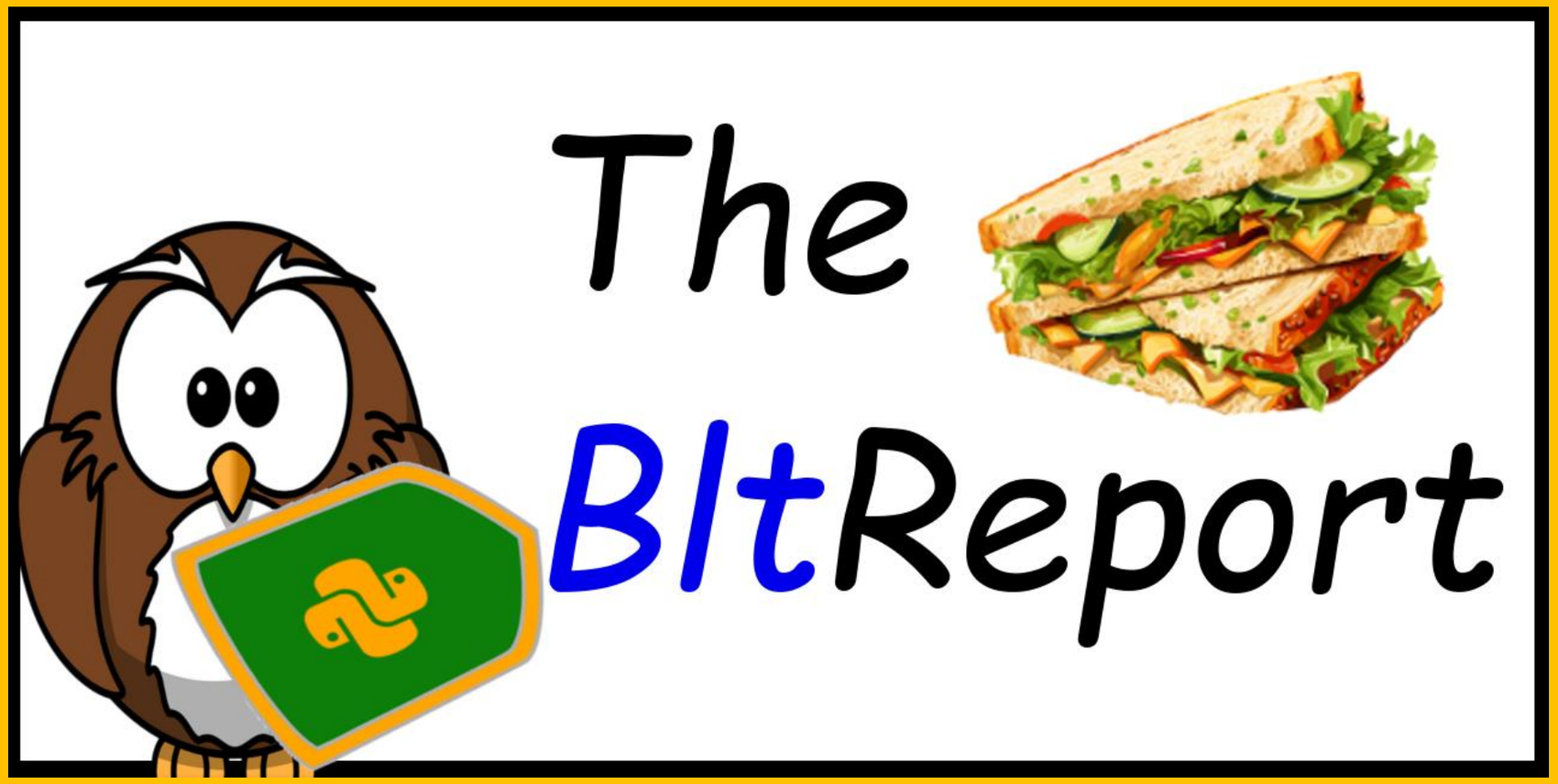
KA3058: Frozenset

Advanced

A frozenset()

(1) Cannot be changed
(2) Can be created from an iterable
(3) Is not a child of set()
(4) Is a built-in function
(5) All of the above

Video: BLT_01300



- False
- all
- bool
- callable
- complex
- dict
- eval
- float
- globals
- hex
- isinstance
- license
- max
- object
- pow
- range
- set
- staticmethod
- tuple
- None
- any
- breakpoint
- chr
- copyright
- dir
- exec
- format
- hasattr
- id
- issubclass
- list
- memoryview
- oct
- print
- repr
- setattr
- str
- type
- True
- ascii
- bytearray
- classmethod
- credits
- divmod
- exit
- frozenset
- hash
- input
- iter
- locals
- min
- open
- property
- reversed
- slice
- sum
- vars
- abs
- bin
- bytes
- compile
- delattr
- enumerate
- filter
- getattr
- help
- int
- len
- map
- next
- ord
- quit
- round
- sorted
- super
- zip

Classic dict()

```
>>> Rpt.type_report(dict)
01 clear        02 copy         03 fromkeys     04 get          05 items
06 keys         07 pop          08 popitem      09 setdefault 10 update
11 values
```

Assignment == ‘Hashable’

- Both dict() & set()

```
>>> dict([[1,2], [3,4], [5,6]])
{1: 2, 3: 4, 5: 6}

>>> {[1,2], [3,4], [5,6]}
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    {[1,2], [3,4], [5,6]}
TypeError: unhashable type: 'list'
```

01 clear

```
>>> help(dict.clear)
Help on method_descriptor:

clear(...)
    D.clear() -> None.  Remove all items from D.
```

02 copy

```
>>> help(dict.copy)
Help on method_descriptor:

copy(...)
    D.copy() -> a shallow copy of D
```

03 fromkeys

```
>>> help(dict.fromkeys)
Help on built-in function fromkeys:

fromkeys(iterable, value=None, /) method of builtins.type instance
    Create a new dictionary with keys from iterable and values set to value.
```

04 get

```
>>> help(dict.get)
Help on method_descriptor:

get(self, key, default=None, /)
    Return the value for key if key is in the dictionary, else default.
```

05 items

```
>>> help(dict.items)
Help on method_descriptor:

items(...)
    D.items() -> a set-like object providing a view on D's items
```

Review: Assignment & Iterating …

```
>>> d = dict([[1,2], [3,4], [5,6]])
>>> for k in d:
        print(f'*d[{k}] is {d[k]}')


*d[1] is 2
*d[3] is 4
*d[5] is 6
```

Common: Assignment == Updating …

- “Array Style” Access

```
>>> d = dict([[1,2], [3,4], [5,6]])
>>> for k in d:
        d[k] *= 2
        print(f'*d[{k}] is {d[k]}')


*d[1] is 4
*d[3] is 8
*d[5] is 12
>>> d
{1: 4, 3: 8, 5: 12}
```

Common: Item Deletion

```
>>> for k in d:
        if k % 3 == 0:
            del d[k]
            continue
        print(f'*d[{k}] is {d[k]}')


*d[1] is 2
*d[5] is 6
>>> d
{1: 2, 5: 6}
```

06 keys

```
>>> help(dict.values)
Help on method_descriptor:

values(...)
    D.values() -> an object providing a view on D's values
```

```
>>> type(dict().values())
<class 'dict_values'>
```

11 values

```
>>> help(dict.keys)
Help on method_descriptor:

keys(...)
    D.keys() -> a set-like object providing a view on D's keys
```

```
>>> type(dict().keys())
<class 'dict_keys'>
```

07 pop

```
>>> help(dict.pop)
Help on method_descriptor:

pop(...)
    D.pop(k[,d]) -> v, remove specified key and return the corresponding value.

    If key is not found, default is returned if given, otherwise KeyError is raised
```

08 popitem

```
>>> help(dict.popitem)
Help on method_descriptor:

popitem(self, /)
    Remove and return a (key, value) pair as a 2-tuple.

    Pairs are returned in LIFO (last-in, first-out) order.
    Raises KeyError if the dict is empty.
```

09 setdefault

```
>>> help(dict.setdefault)
Help on method_descriptor:

setdefault(self, key, default=None, /)
    Insert key with a value of default if key is not in the dictionary.

    Return the value for key if key is in the dictionary, else default.
```

10 update

```
>>> help(dict.update)
Help on method_descriptor:

update(...)
    D.update([E, ]**F) -> None.  Update D from dict/iterable E and F.
    If E is present and has a .keys() method, then does:  for k in E: D[k] = E[k]
    If E is present and lacks a .keys() method, then does:  for k, v in E: D[k] = v
    In either case, this is followed by: for k in F:  D[k] = F[k]
```

Dictionary Comprehensions?

- Worthless, or Wonderful?

```
>>> {ss:val for ss, val in enumerate('results') if val in 'aeiou'}
{1: 'e', 3: 'u'}
```

KA1013: Key-Value Pairs

Beginner

What is the difference between set and dict?

(1) dict manages key value pairs
(2) set can have duplicate values
(3) dict can have duplicate keys
(4) set cannot be reverse-sorted
(5) set and dict are equivalent

KA1014: Sorting Dictionaries

Beginner

How can we custom sort dictionary values?

(1) Use sorted() with a 'key' function
(2) Use sorted() with a 'filter' function
(3) Use the .sort() operation
(4) Use the .sorted() operation
(5) Use filter() with a 'sort' operation

KA1015: Dictionary Orderings

Beginner

Does dict() insure insertion-key ordering?

(1) dict() manages ordering
(2) dict() never manages ordering
(3) dict() always respects ordering
(4) dict() may / may not preserve order
(5) dict() does not use key value pairs

KA1016: Dictionary Changes

Beginner

How do we add a key:value pair to my_dict?

(1) my_dict.append('key', value)
(2) my_dict.insert('key', value)
(3) my_dict['key'] = value
(4) my_dict[value] = 'key'
(5) None of the above

KA1017: Dictionary Removal

Beginner

How do we remove 'key' from my_dict?

(1) my_dict.kill('key', value)
(2) my_dict.remove('key', value)
(3) my_dict['key'] = None
(4) del my_dict['key']
(5) None of the above

KA3031: Object Management

Advanced

How to check object membership?

(1) Use Inheritance
(2) Use 'Duck Typing'
(3) Use hasattr()
(4) Use getattr()
(5) Either 3 or 4

KA3035: Object Management

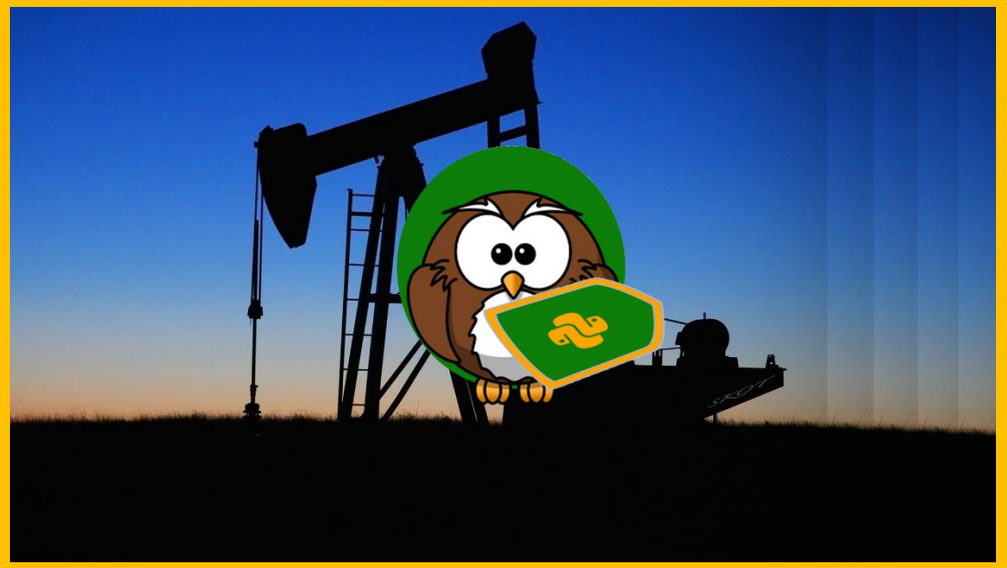
Advanced

Use setattr() to:

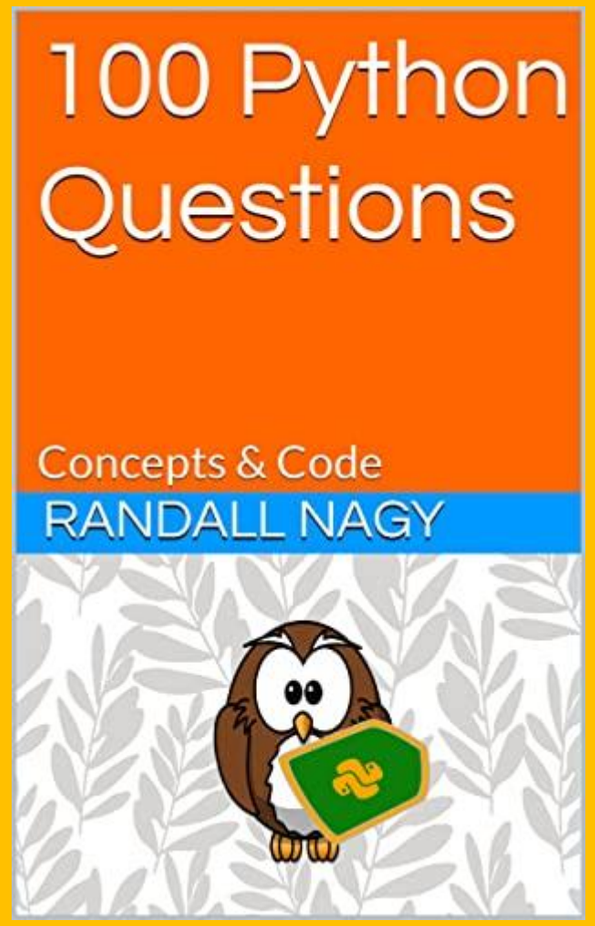
(1) Replace existing member
(2) Assign new attribute
(3) Safely check presence
(4) Attempt attribute removal
(5) Two of the above

Modern Python

Happy PyQuesting!



(presentation end)



Only 27 More To Go!