

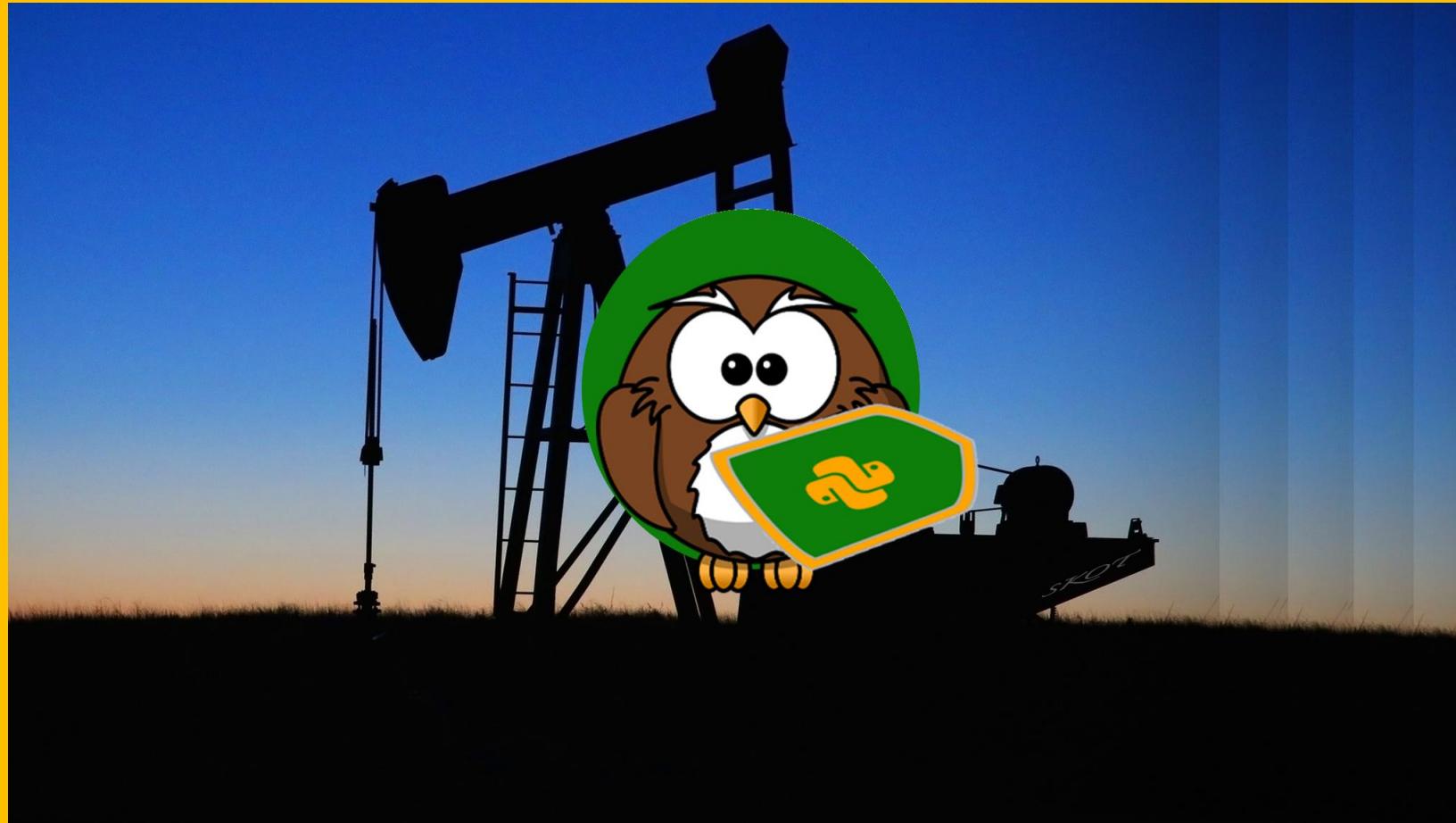
# Modern Pythoneering

# The Built-In Reports

*By Randall Nagy*



# A.K.A: PyQuesting!





# 100 Python Questions

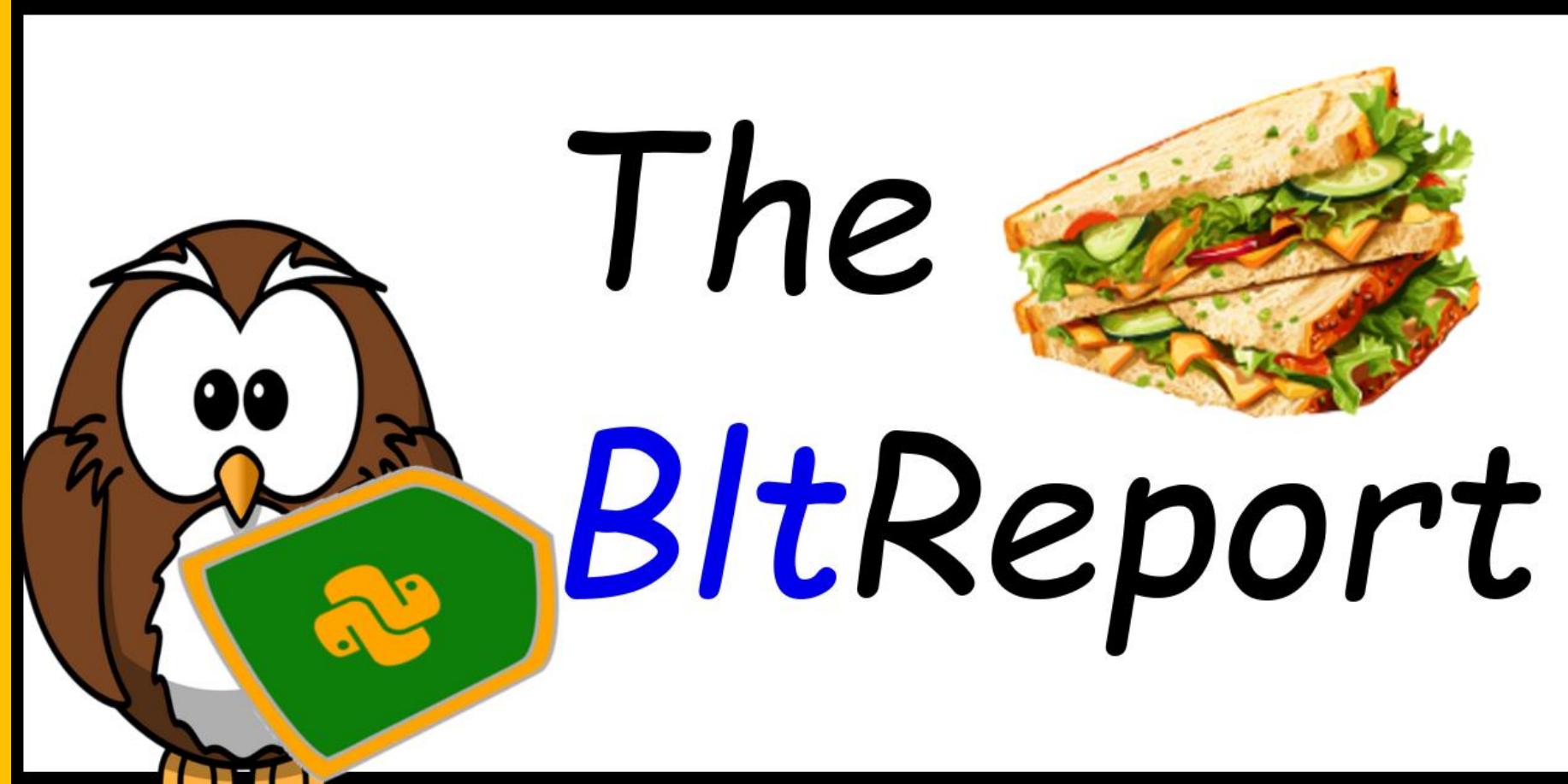
Concepts & Code

RANDALL NAGY





Video: BLT\_00100



# The BitReport



# The ‘Upper-Cased’

Keywords:

- True
- False
- None



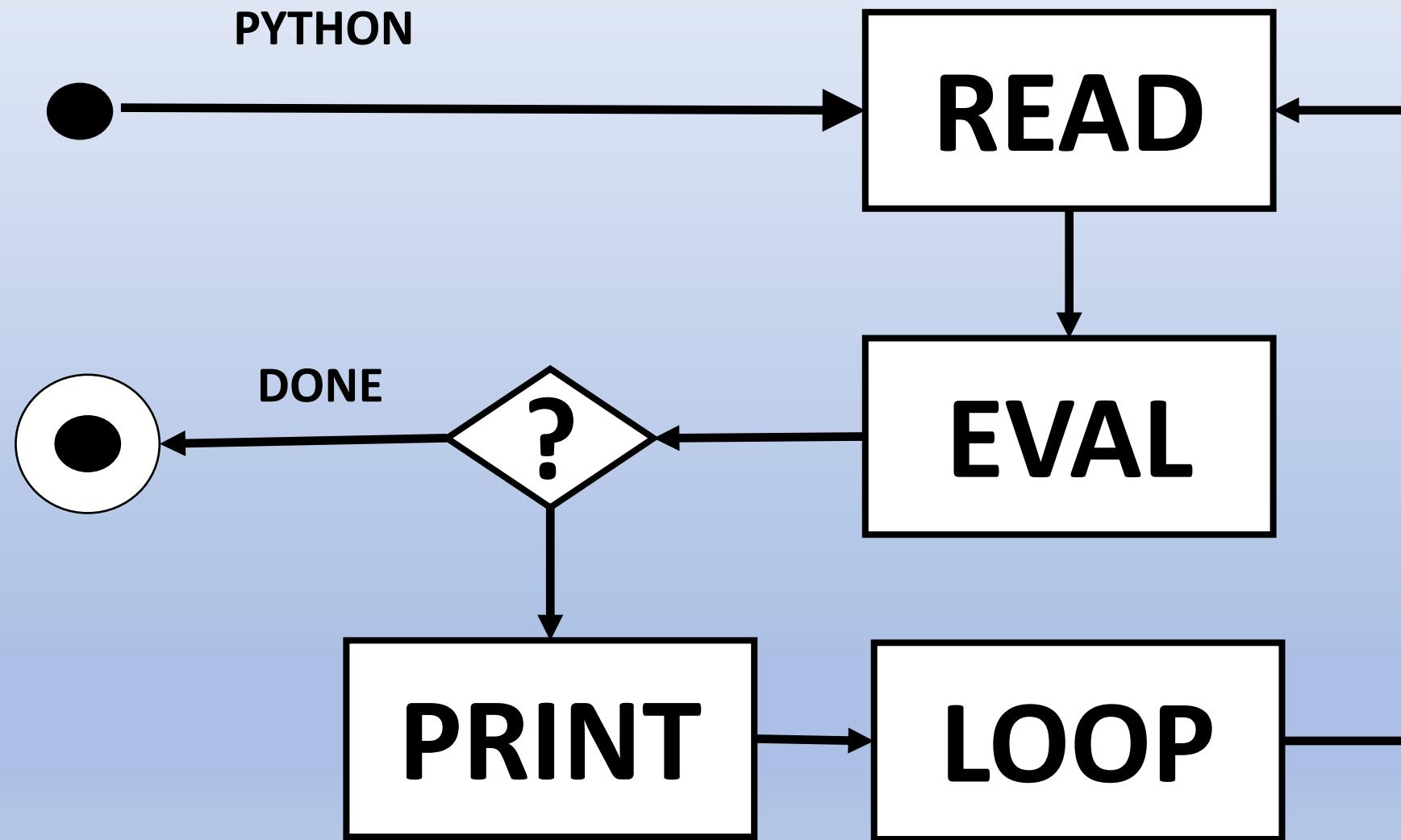
# The Built-Ins

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

1

R0





## 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

1

R0





## 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

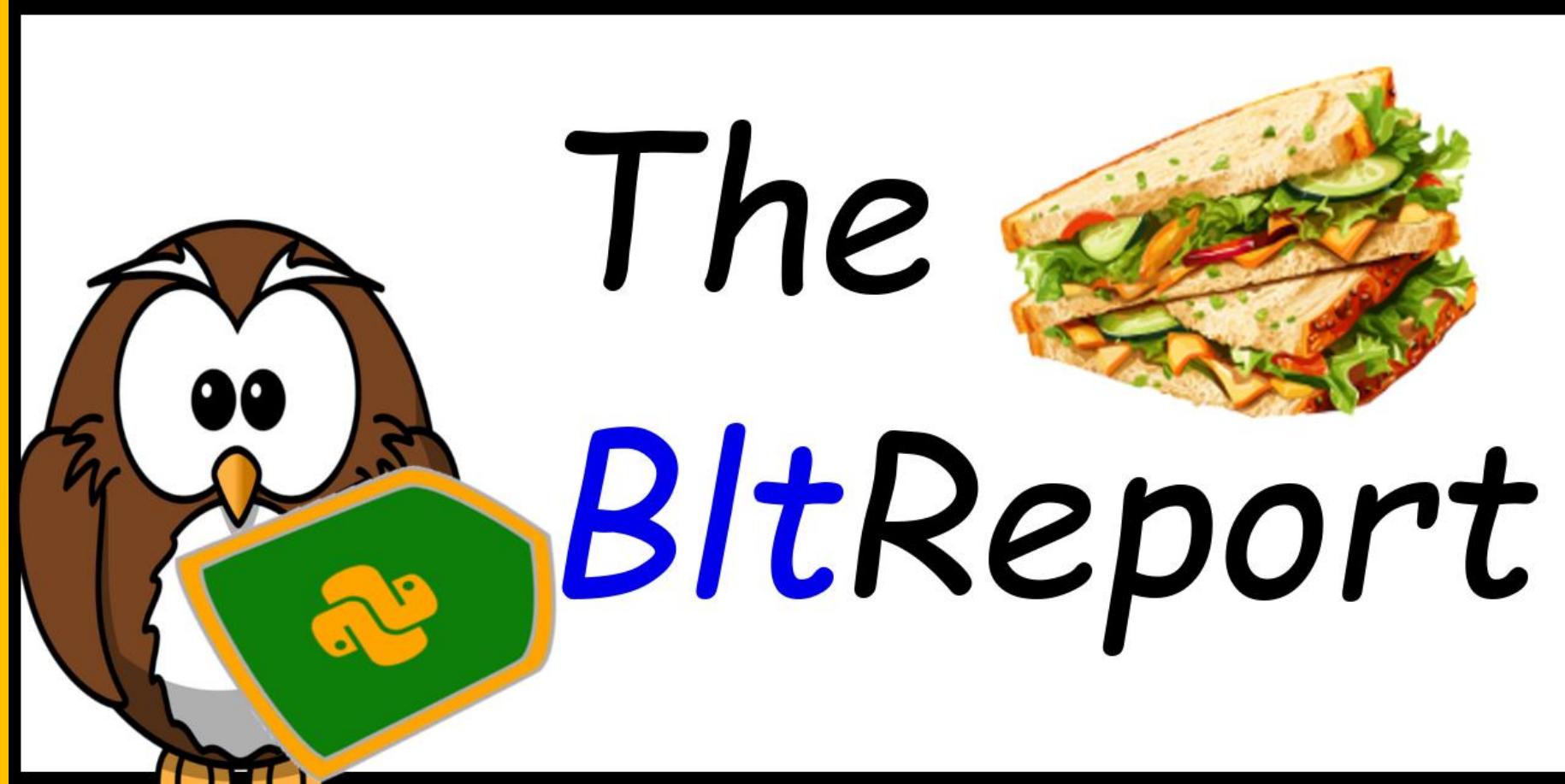
1

R0



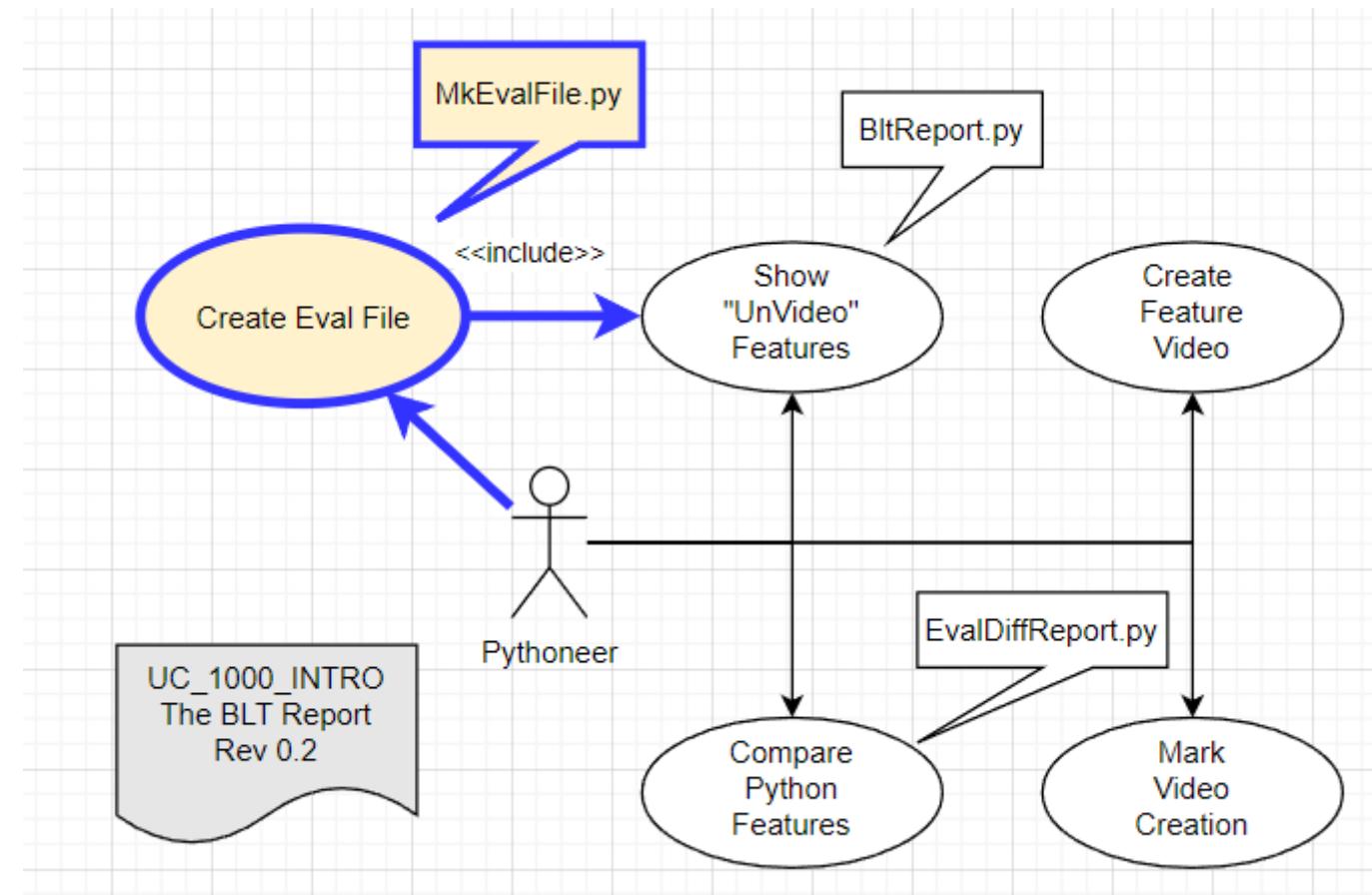


Video: BLT\_00200





# 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

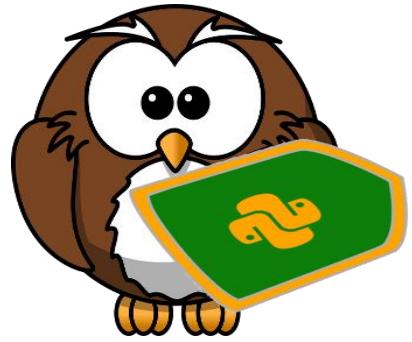
Return integer ratio.

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`

The screenshot shows a window titled "IDLE Shell" with a menu bar: File, Edit, Shell, Debug, Options, Window, Help. The main area contains Python code and its output. The code consists of five lines of Python code starting with '>>>'. The first three lines show that bool(255) and bool(-255) both return True, while bool(0) returns False. The last line shows a blank line starting with '>>>'.

```
>>> bool(255)
True
>>> bool(-255)
True
>>> bool(0)
False
>>> |
```

Ln: 31 Col: 4



# 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**
- ▷ **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**
- ▶ **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**



## KA1061: Integer Values

Beginner

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

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

1

R0





## 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

1

R0





## KA2037: Boolean Values

### Intermediate

```
>>> bool (-255)
```

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

1

R0





## 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

1

R0





## 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

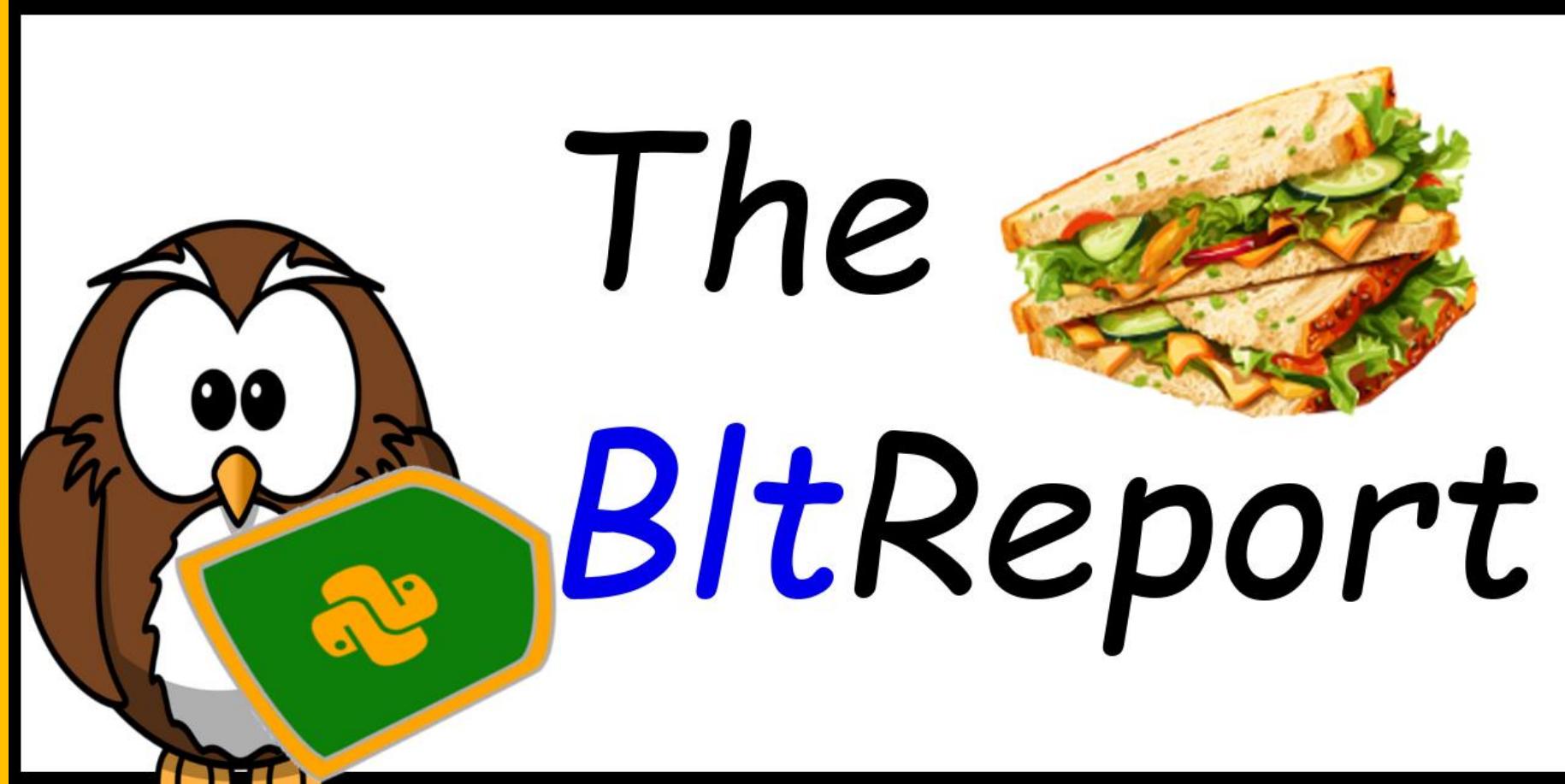
1

R0





Video: BLT\_00300



# The BitReport

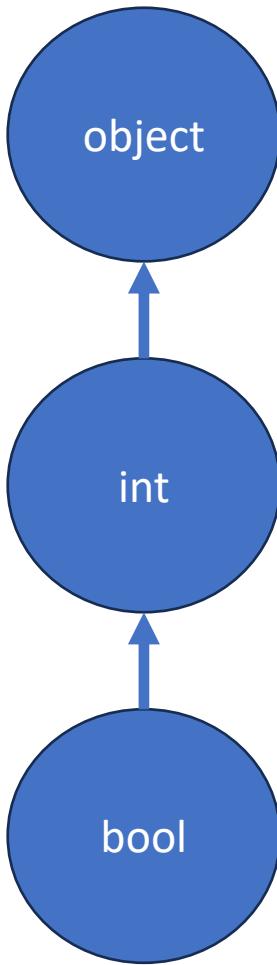


- ▶ **False**
- ▷ **all**
- ▶ **bool**
- ▷ **callable**
- ▷ **complex**
- ▷ **dict**
- ▶ **eval**
- ▷ **float**
- ▷ **globals**
- ▷ **hex**
- ▷ **isinstance**
- ▶ **license**
- ▷ **max**
- ▷ **object**
- ▷ **pow**
- ▷ **range**
- ▷ **set**
- ▷ **staticmethod**
- ▷ **tuple**
- ▶ **None**
- ▷ **any**
- ▷ **breakpoint**
- ▷ **chr**
- ▶ **copyright**
- ▷ **dir**
- ▷ **exec**
- ▷ **format**
- ▷ **hasattr**
- ▷ **id**
- ▷ **issubclass**
- ▷ **list**
- ▷ **memoryview**
- ▷ **oct**
- ▶ **print**
- ▷ **repr**
- ▷ **setattr**
- ▷ **str**
- ▶ **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**



# '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

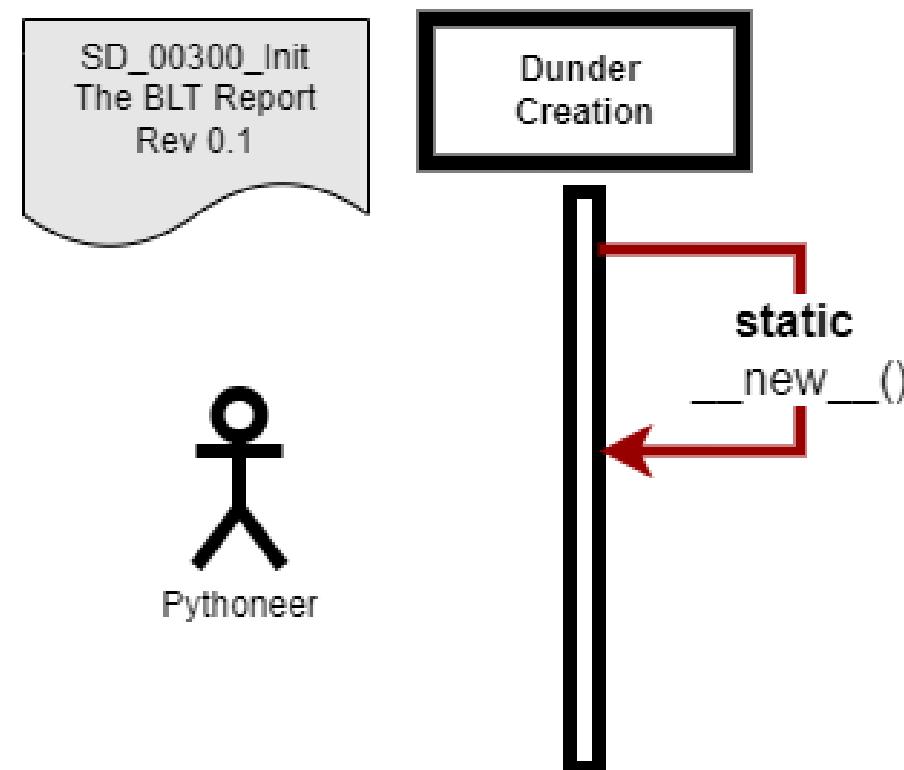
- Case Study: BltTypeEx.py

```
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 ...

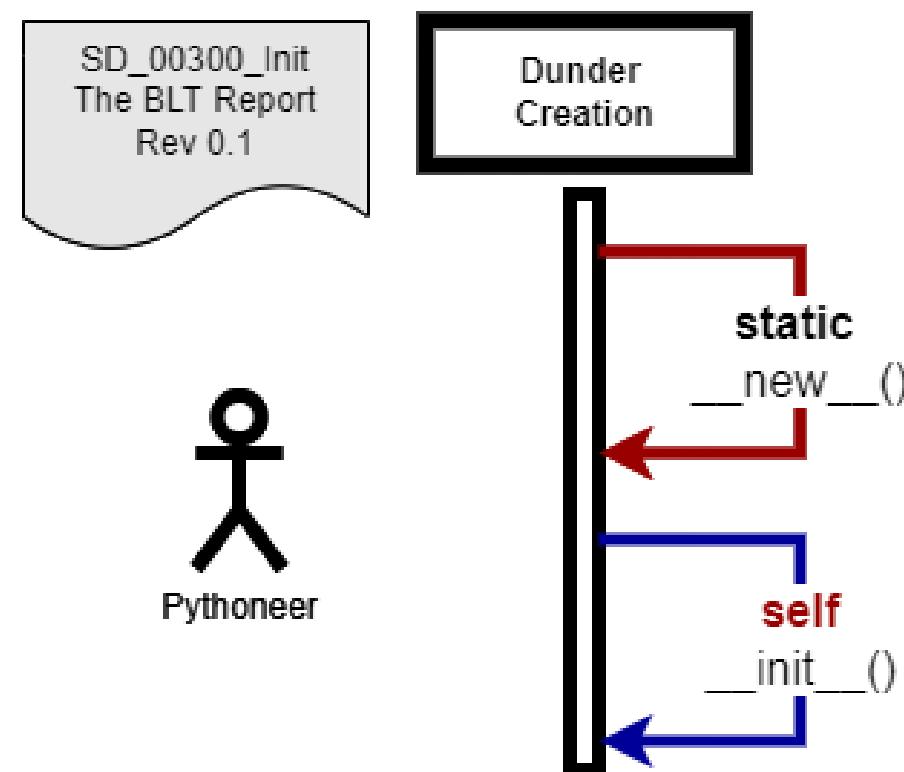
- Case Study: BltTypeEx.py

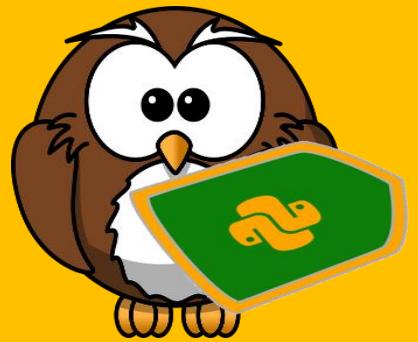




# Review: Classic Initialization

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



## KA2039: Instance Detection

Intermediate

>>> isinstance()

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

1

R0





## KA2040: Class Detection

Intermediate

>>> issubclass()

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

1

R0





## 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

1

R0





## 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

1

R0





## 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

1

R0





## 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

1

R0





## 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

1

R0





## 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

1

R0





## 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

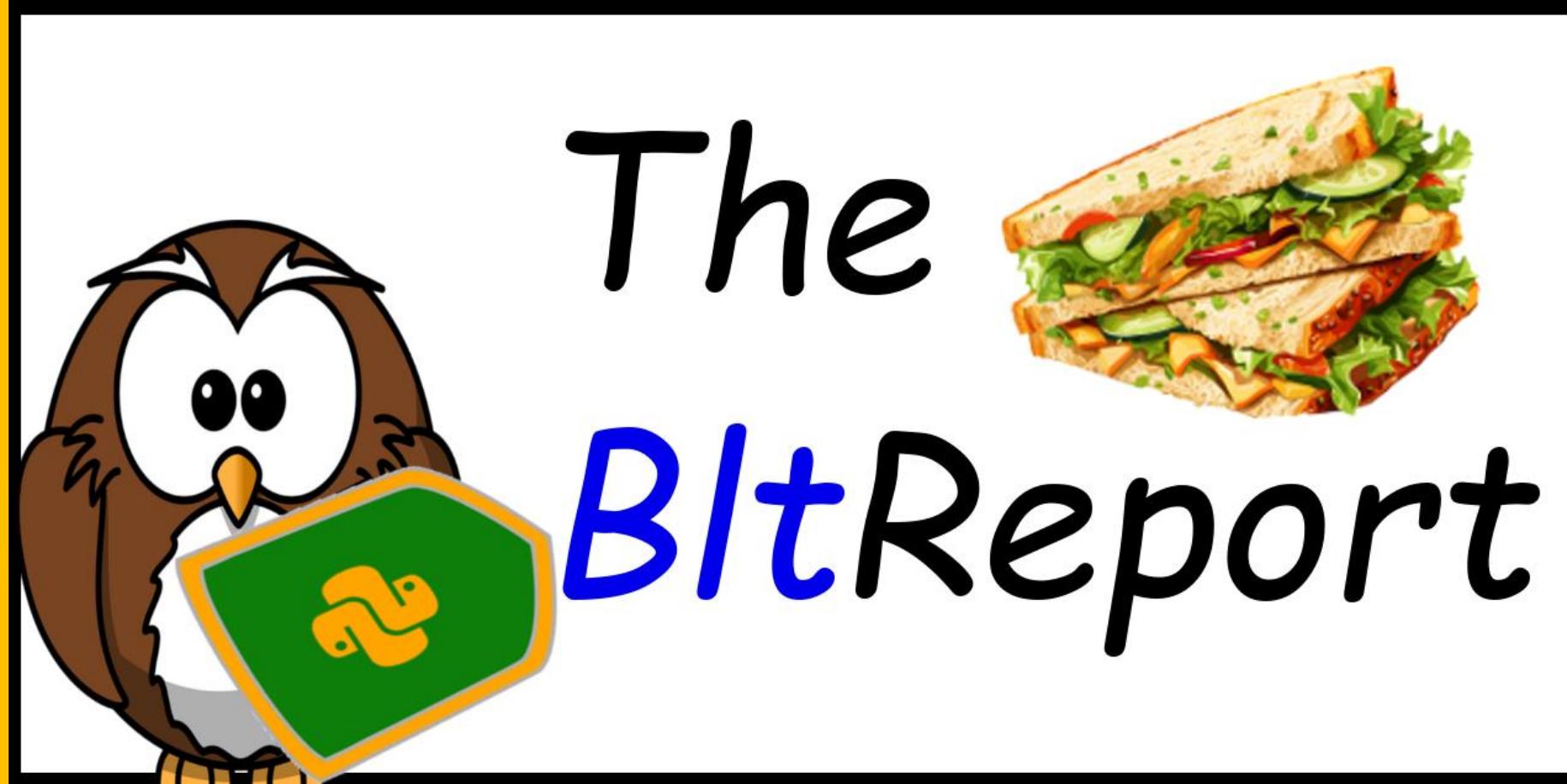
1

R0





Video: BLT\_00400



# The BitReport



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



# 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**
- ▷ **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**



## KA1063: Enumerations

Beginner

Built-in enumerate()

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

1

R0





## KA1064: Iteration

Beginner

Which types are iterable?

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

1

R0





## KA1065: Iteration

Beginner

Which types are iterable?

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

1

R0





## 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

1

R0





## KA3040: Iteration Management

Advanced

Iterables must implement `__next__`

- (1) True
- (2) False

1  
R0





## KA3041: Iteration Management

Advanced

Iterables must implement `__iter__`

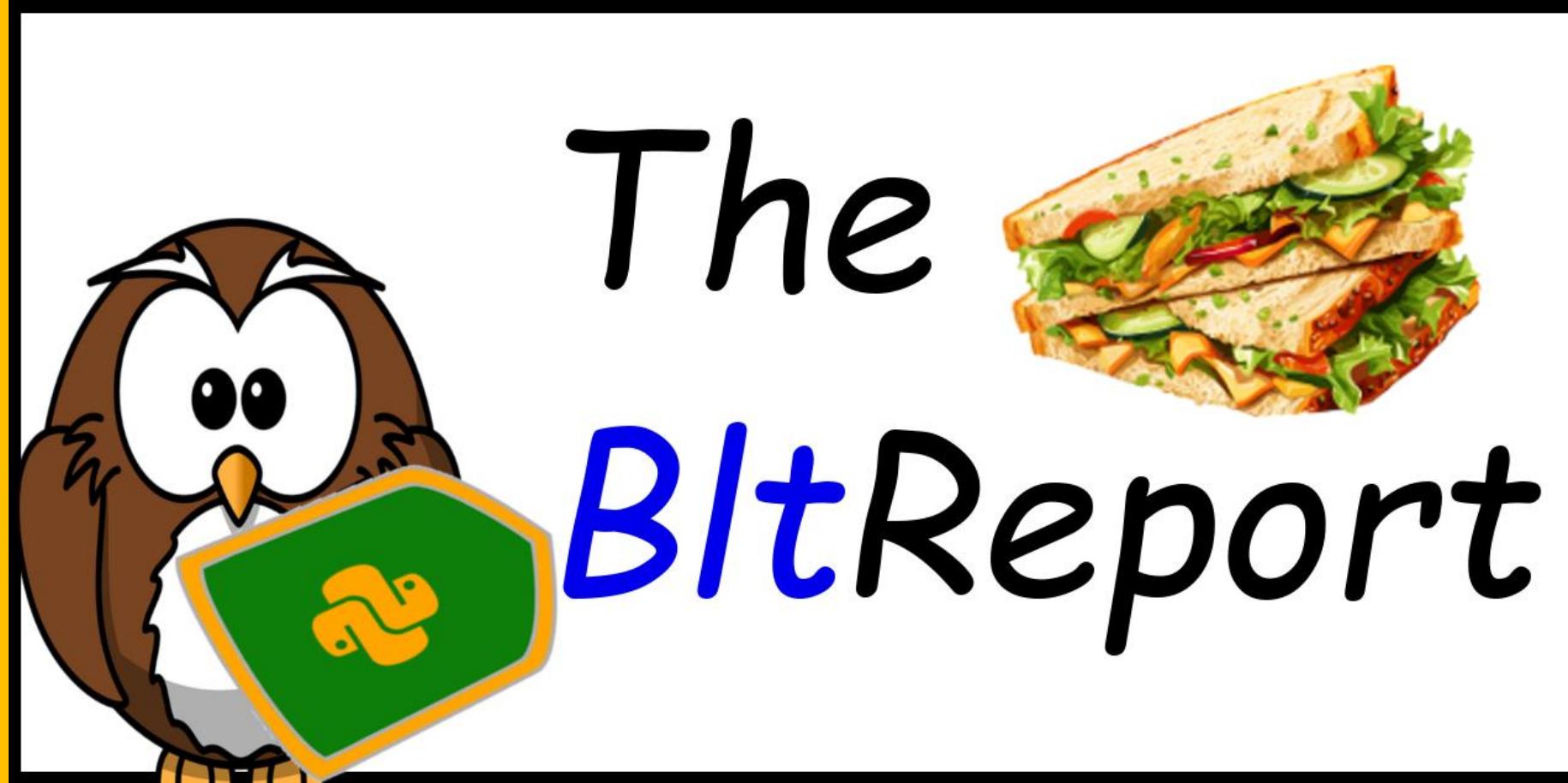
- (1) True
- (2) False

1  
R0





Video: BLT\_00500



# The BitReport



# Focus: range(...)

- ▶ `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`



# 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	<code>__bool__</code>	03	<code>__getitem__</code>
02	<code>__contains__</code>	05	<code>__len__</code>
04	<code>__iter__</code>	07	<code>count</code>
06	<code>__reversed__</code>	09	<code>start</code>
08	<code>index</code>	11	<code>stop</code>
10	<code>step</code>		



# Commons: Range & Enumerate

```
1  r = set(dir(range(5)))    __  __  __  __  
2  e = set(dir(enumerate(''))) __  __  __  __  
3  z = sorted(r.intersection(e))  
4  for ss, item in enumerate(z):  
5      if ss % 2 == 0:  
6          print()  
7          print(f'{ss+1:>02} {item:<20}', end=' ')  
8  __ __  
9  __ __  
10 __ __  
11 __ __  
12 __ __  
13 __ __  
14 __ __  
15 __ __  
16 __ __  
17 __ __  
18 __ __  
19 __ __  
20 __ __  
21 __ __  
22 __ __  
23 __ __  
24 __ __
```



## 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

1

R0





## 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

1

R0





## 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

1

R0





## 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

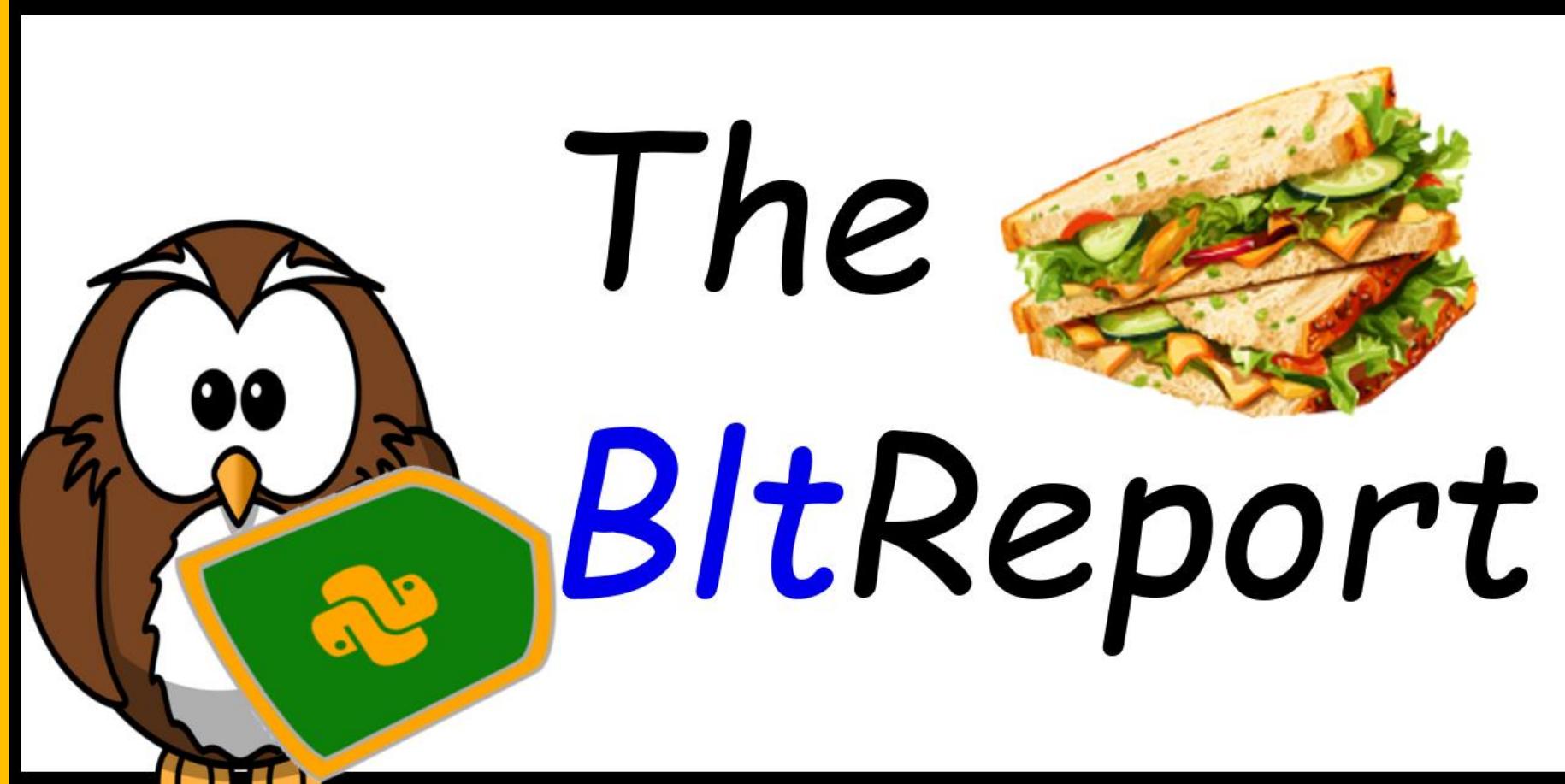
1

R0





Video: BLT\_00600



# The BitReport



- ▶ **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

1

R0





## 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

1

R0





## 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

1

R0





## 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

1

R0





## 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()

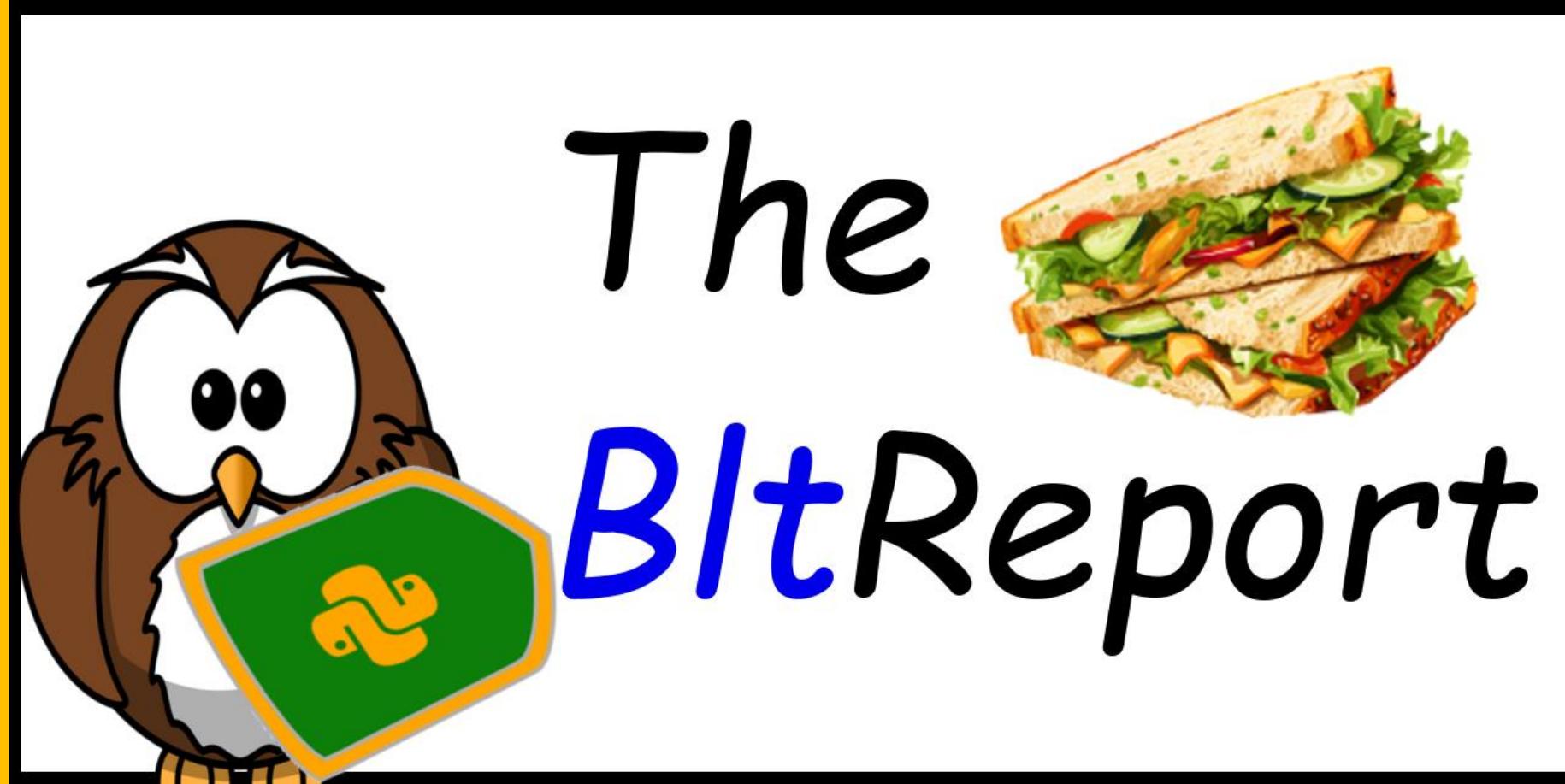
1

R0





Video: BLT\_00700





- ▶ **False**
- ▶ **None**
- ▶ **True**
- ▶ **abs**
- ▶ **all**
- ▶ **any**
- ▶ **ascii**
- ▶ **bin**
- ▶ **bool**
- ▶ **breakpoint**
- ▶ **bytarray**
- ▶ **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 global and local mappings.
```

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 global and local mappings.
```

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.
```

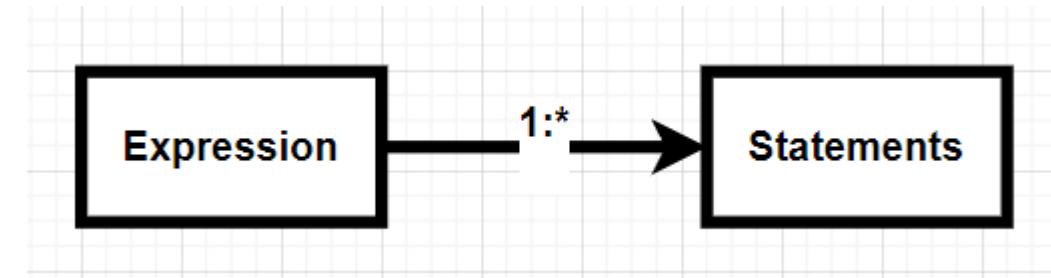
NOTE: Whether or not updates to this dictionary will affect name lookups in the local scope and vice-versa is **\*implementation dependent\*** and not covered by any backwards compatibility guarantees.

→ 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 Evaluation

### 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

1

R0





## 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

1

R0





## KA3045: Globals

Advanced

Pythons built-in global() returns a

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

1

R0





## KA3046: Expressions -v- Statements

Advanced

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

- (1) True
- (2) False

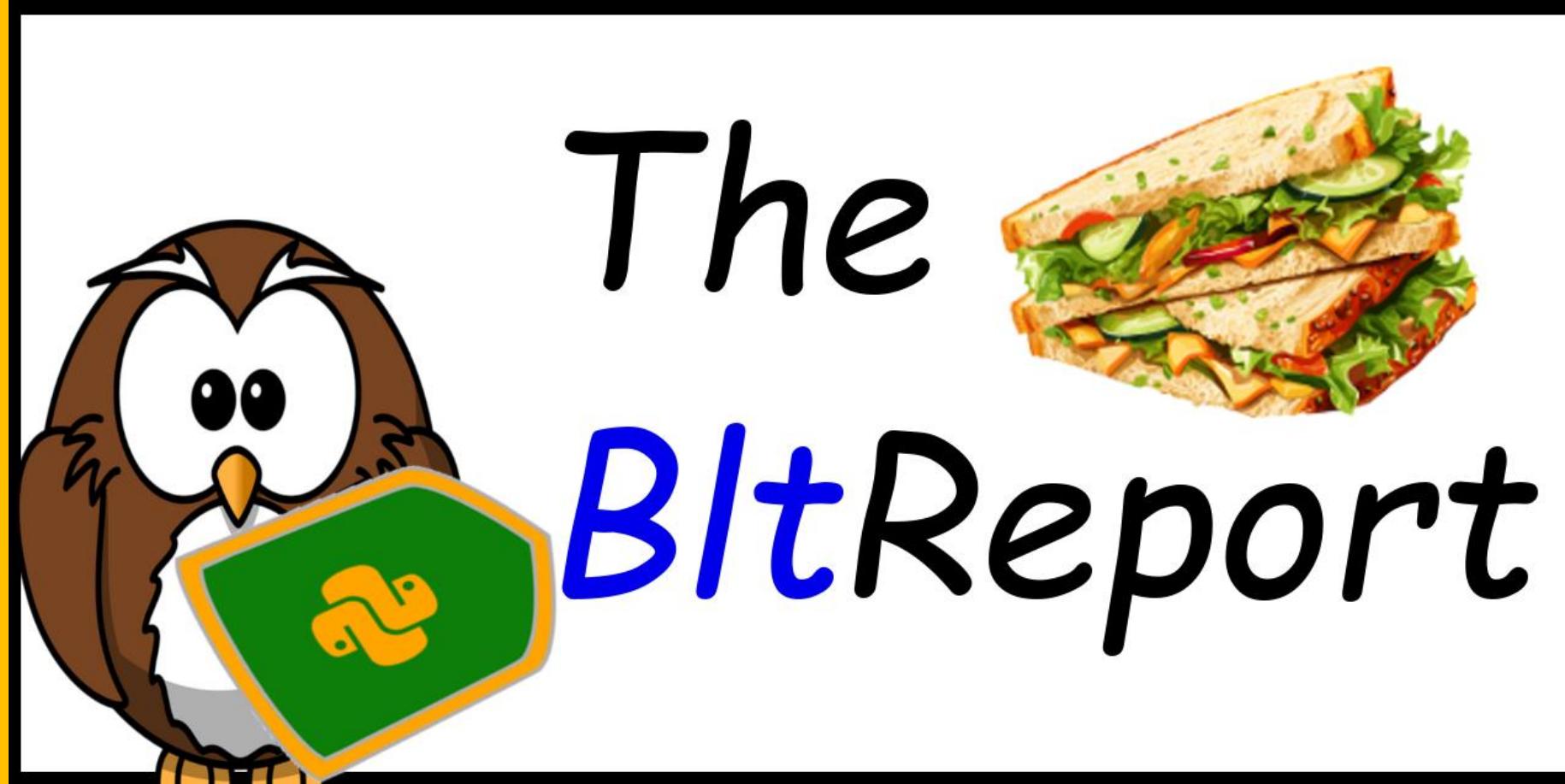
1

R0





Video: BLT\_00800



# The BitReport



- ▶ [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](#)



# Info: compile()

```
>>> help(compile)
```

```
Help on built-in function compile in module builtins:
```

```
compile(source, filename, mode, flags=0, dont_inherit=False, optimize=-1, *, _feature_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'

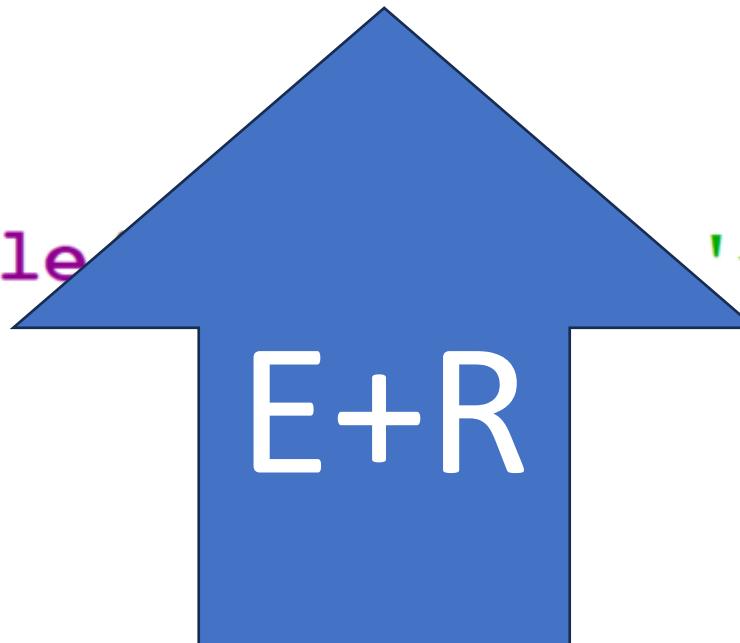
```
>>> a=4;eval(compile('a+5','<string>','eval'))
9
>>> a
4

>>> 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('a*=10;a+5','<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’

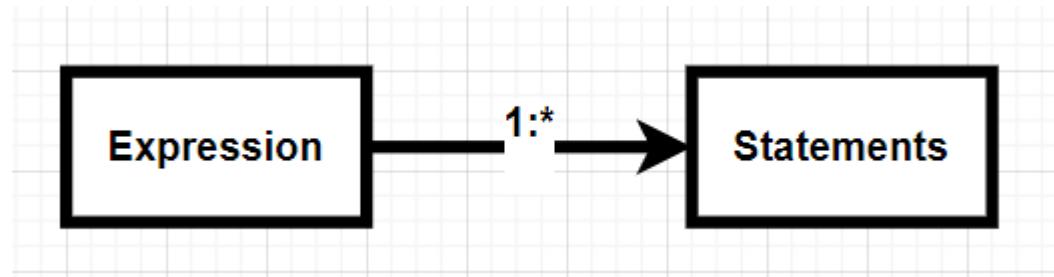
- Compile an Expression
    - Compile a Module
    - Effects & Results

- ‘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’, ‘execute’
- 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

1

R0





## 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

1

R0





## 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

1

R0





## 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

1

R0





## 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

1

R0





## KA3051: Compile

Advanced

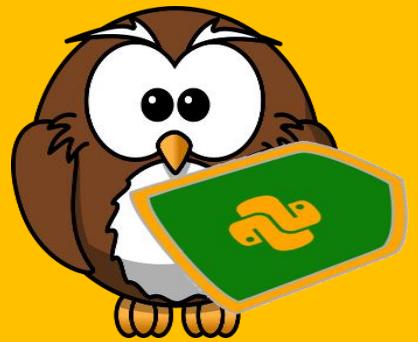
```
a=4;eval(compile('a+=5','','eval'))
```

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

1

R0





# Modern Python

Happy PyQuesting!



(presentation end)

100 Python  
Questions

Concepts & Code  
RANDALL NAGY

