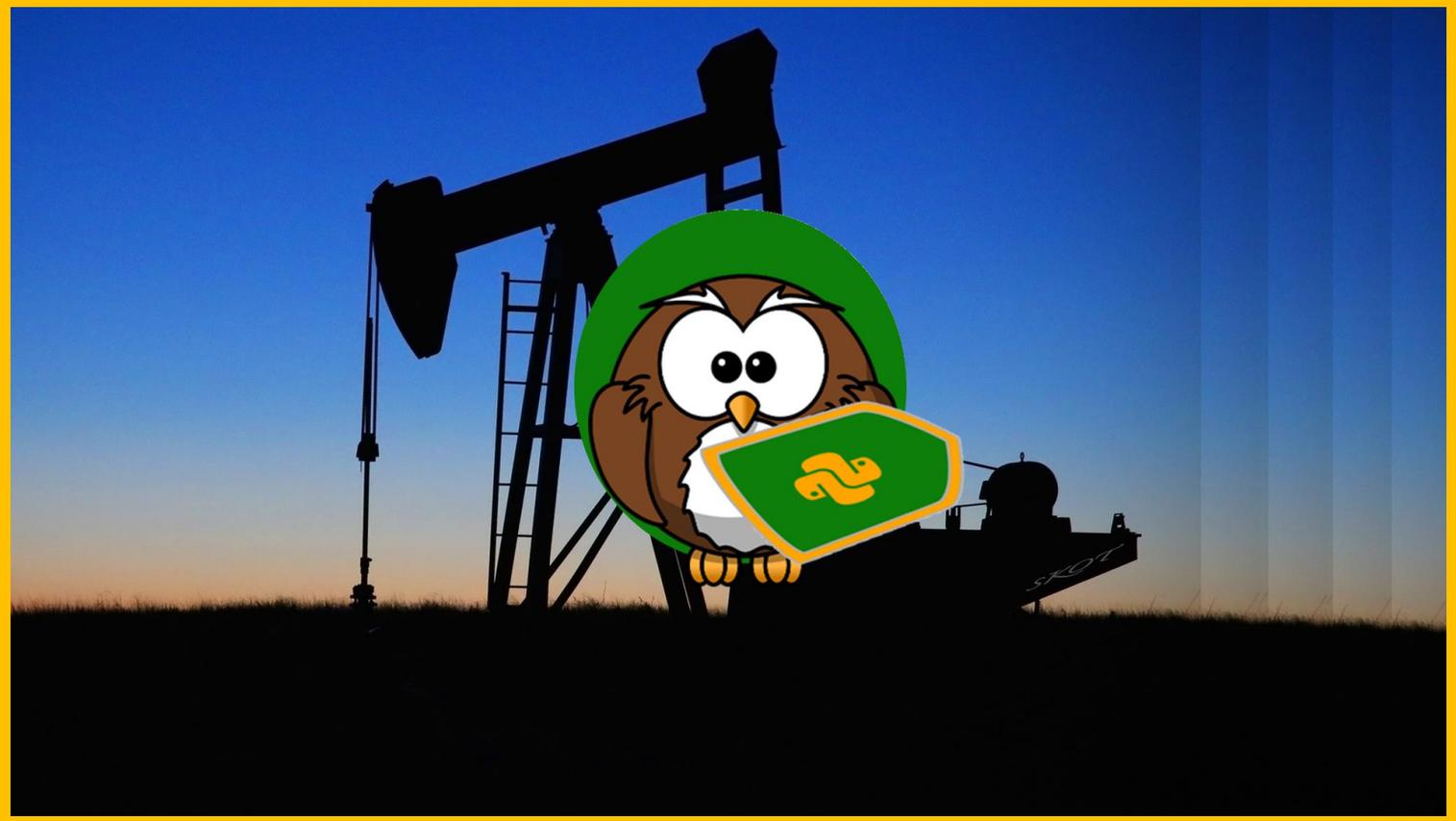
Modern Pythoneering

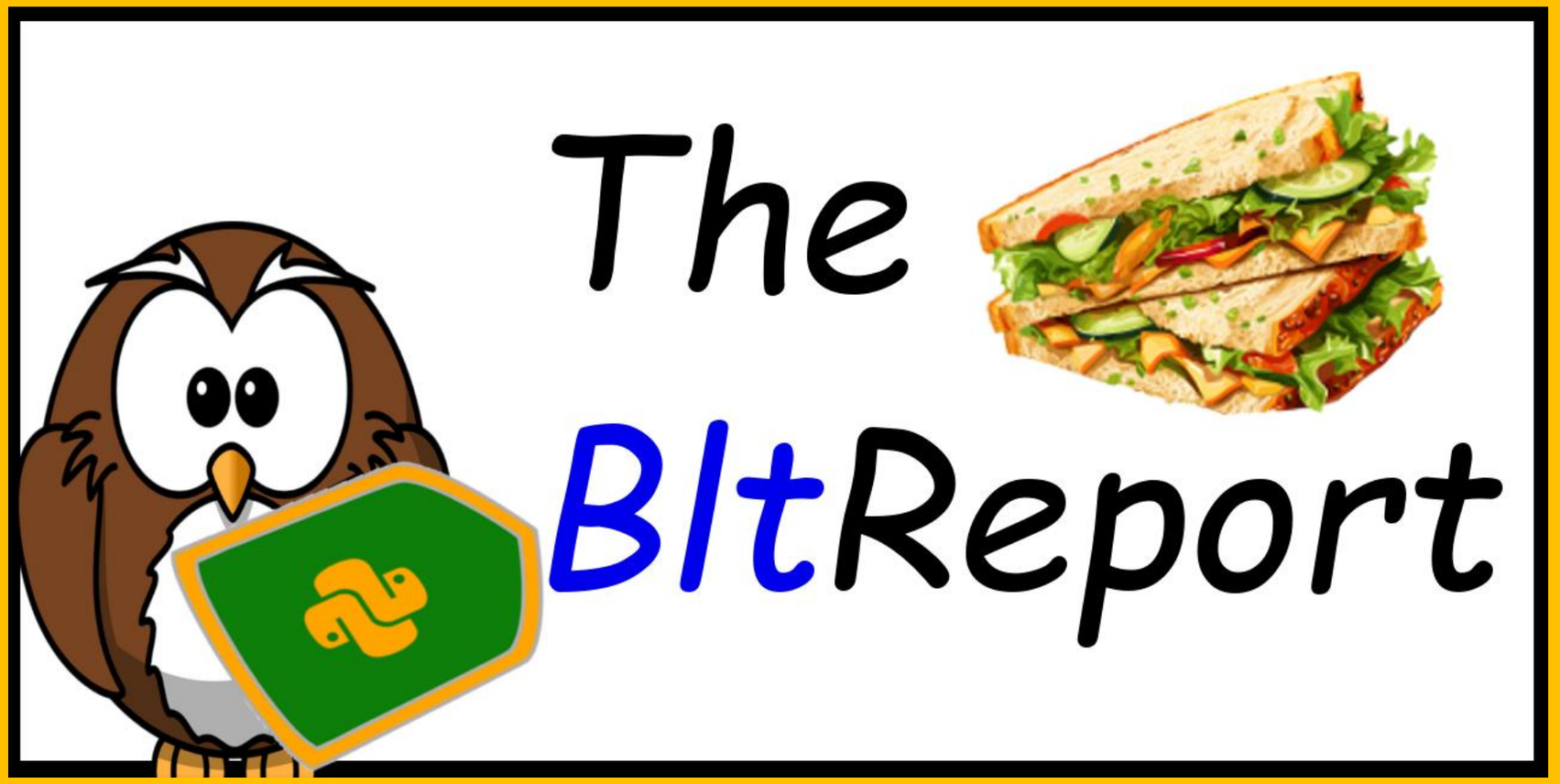
The Built-In Reports

By Randall Nagy

A.K.A: PyQuesting!



Video: BLT_00100



The ‘Upper-Cased’

Keywords:

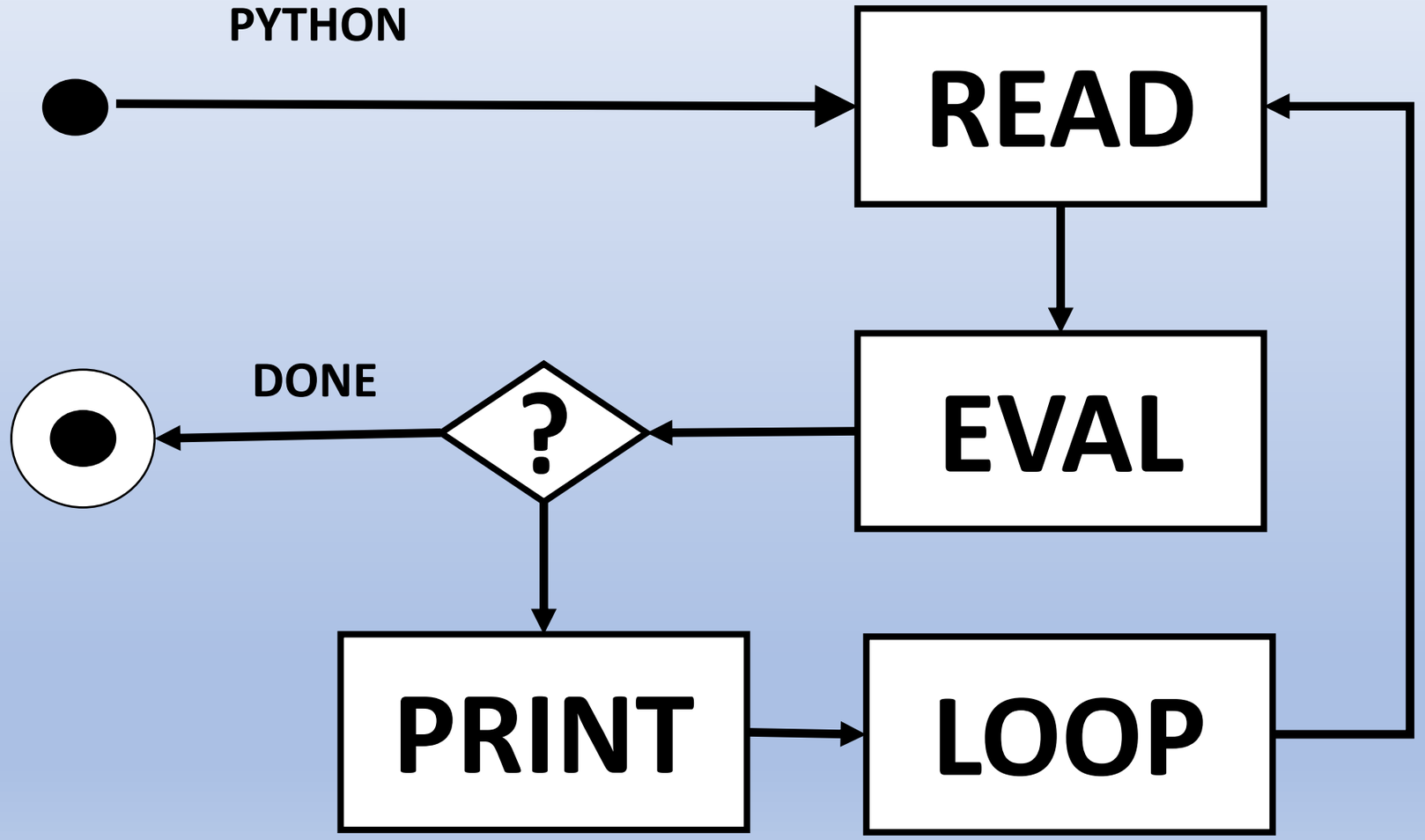
- True
- False
- None

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

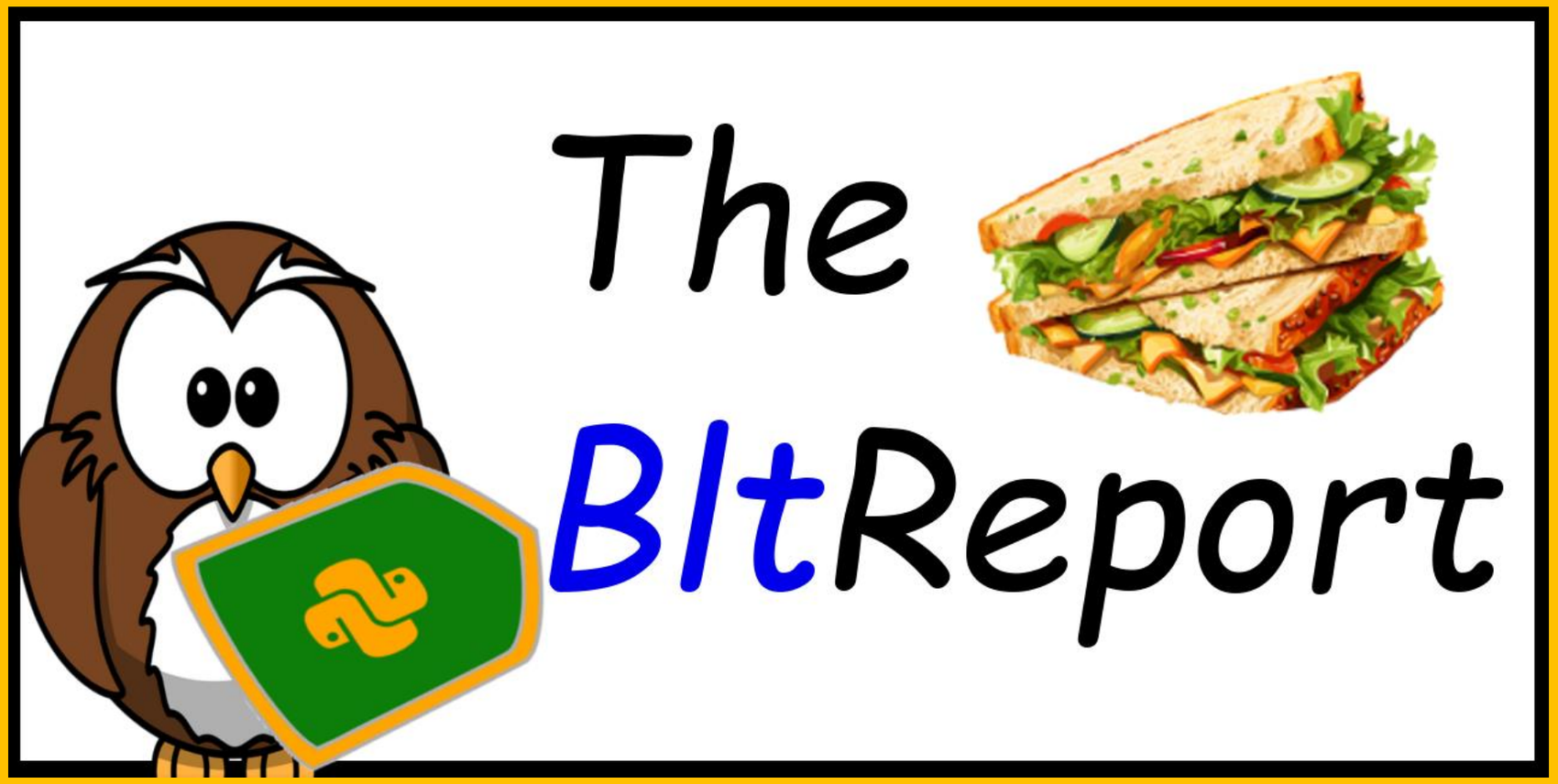
KA1060: Evaluations

Beginner

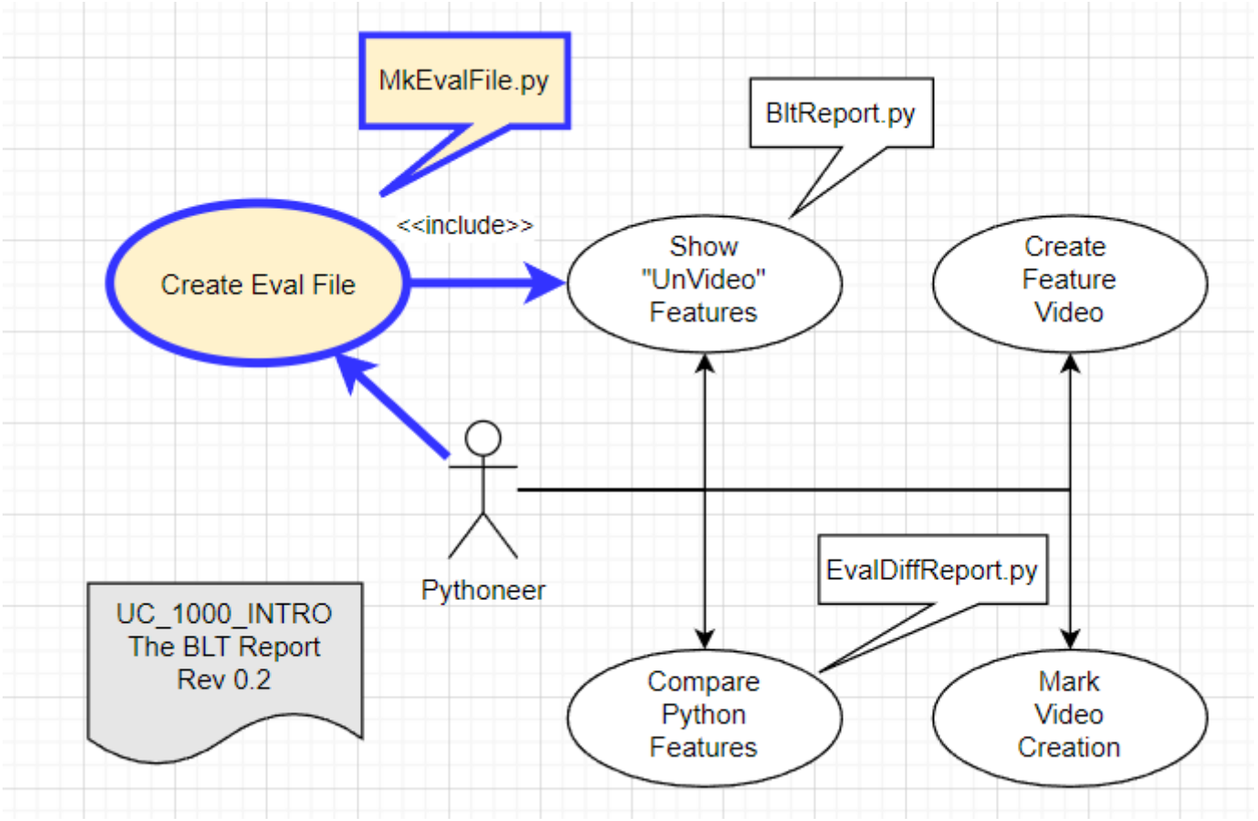
```
eval('bool(1)') will:
```

(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

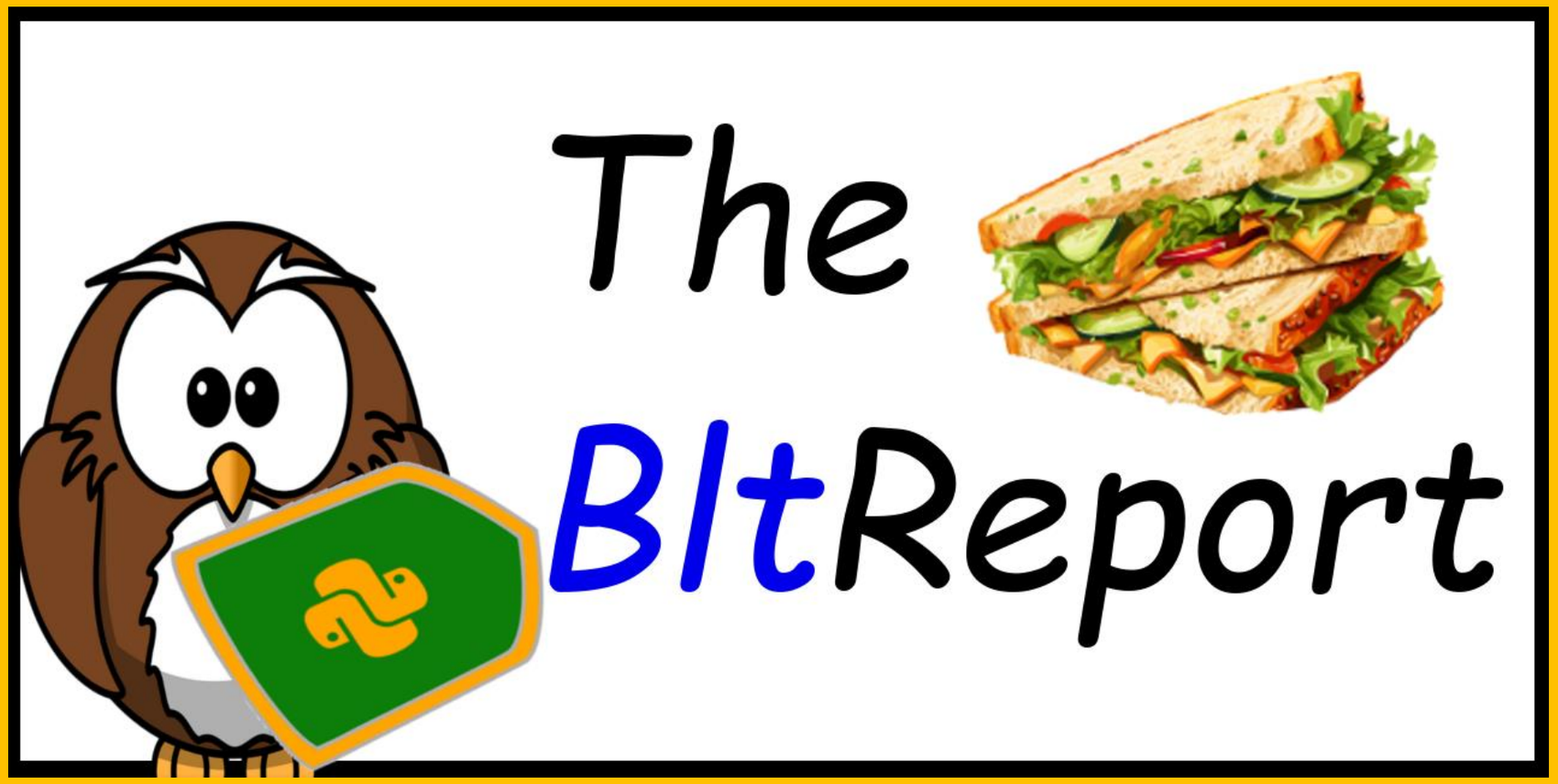
KA3036: Type Management

Advanced

Use type() to:

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

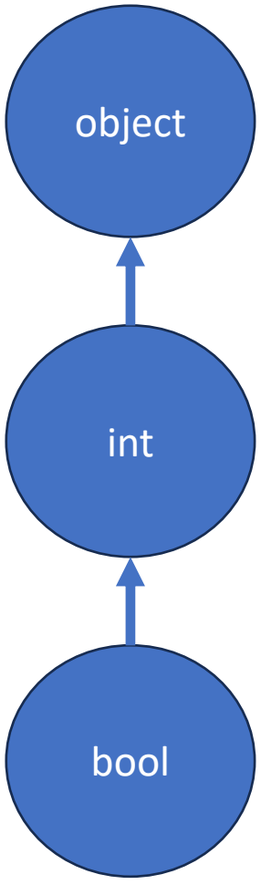
Video: BLT_00300



▶ 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(True, int)
  - issubclass(7, bool)

Review: type() Initialization

- Case Study: BltTypeEx.py

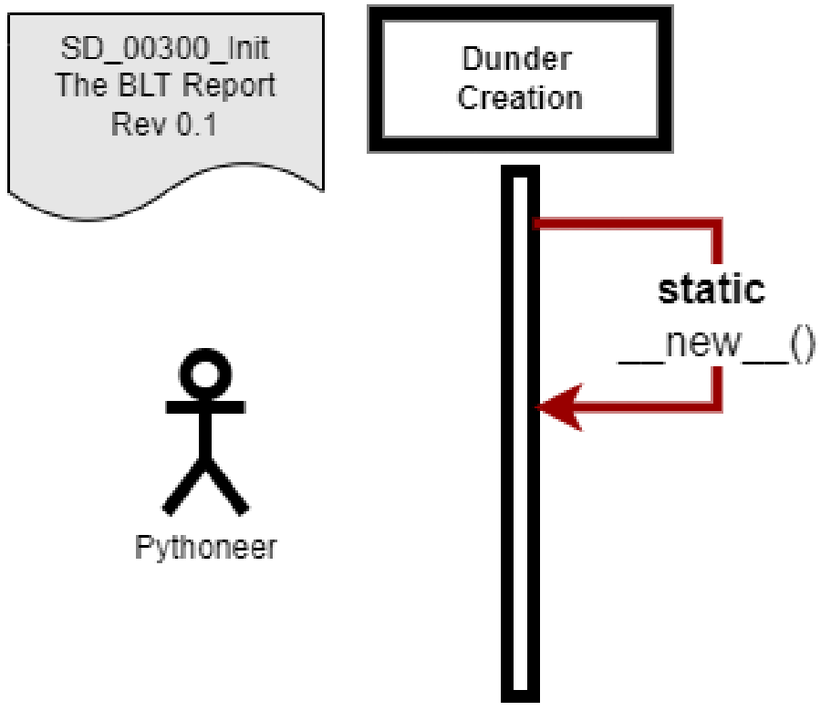
```
class zclass:
    def __init__(self, **kwargs):
        self.times = 1000
        print(f'Created zclass {kwargs}')

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

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

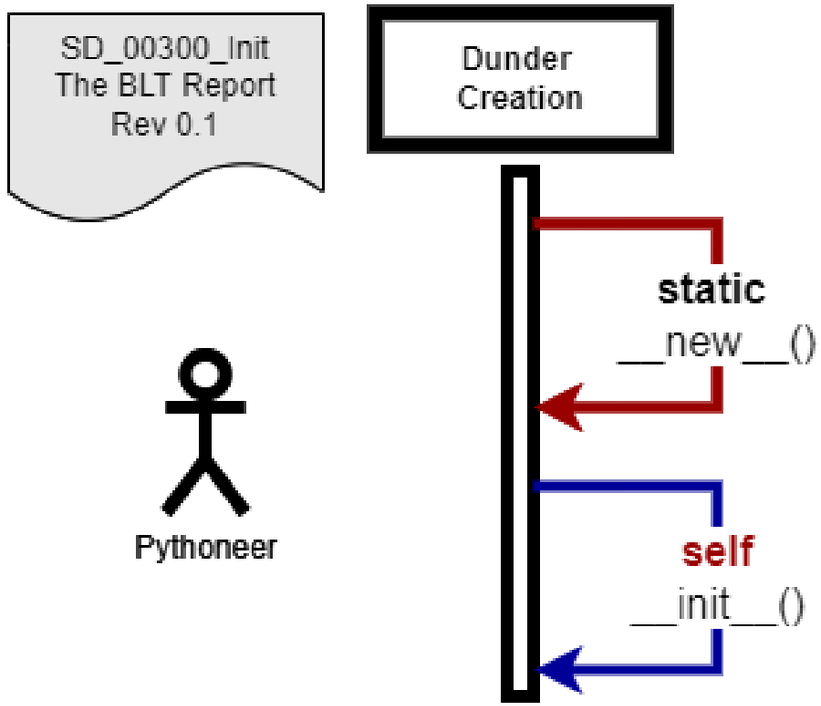
Review: Classic Initialization …

- Case Study: BltTypeEx.py



Review: Classic Initialization

- Case Study: BltTypeEx.py



The “Four ‘Atters”

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

Built-In 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()
>>> print(setattr(z, 'a', True))
None
```

Built-In hasattr()

```
>>> 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.

>>> class Z:
        pass

>>> z.a = True
>>> hasattr(z, 'a')
True
```

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.

>>> class Z:
        pass

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

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

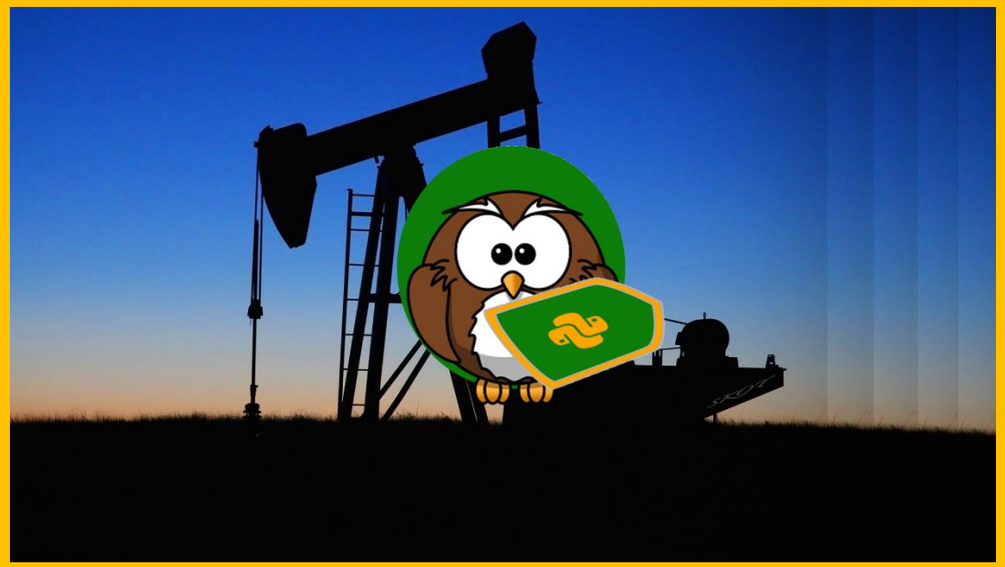
- ▶ None
- ▷ any
- ▷ breakpoint
- ▷ chr
- ▶ copyright
- ▷ dir
- ▷ exec
- ▷ format
- ▶ hasattr
- ▷ id
- ▶ issubclass
- ▷ list
- ▷ memoryview
- ▷ oct
- ▶ print
- ▷ repr
- ▶ setattr
- ▷ str
- ▶ type

- ▶ 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

Modern Python

Happy PyQuesting!



(presentation end)