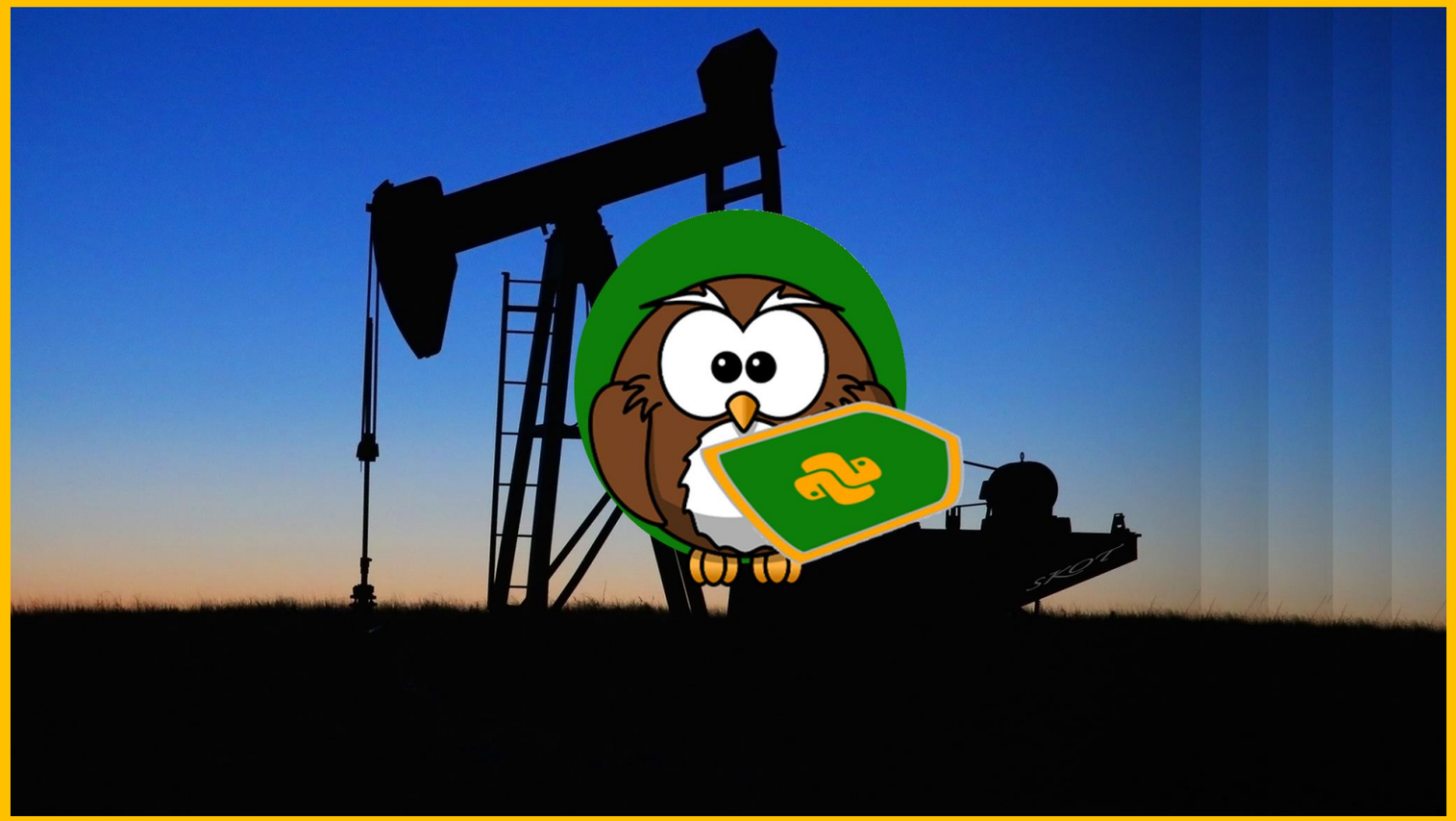
Modern Pythoneering

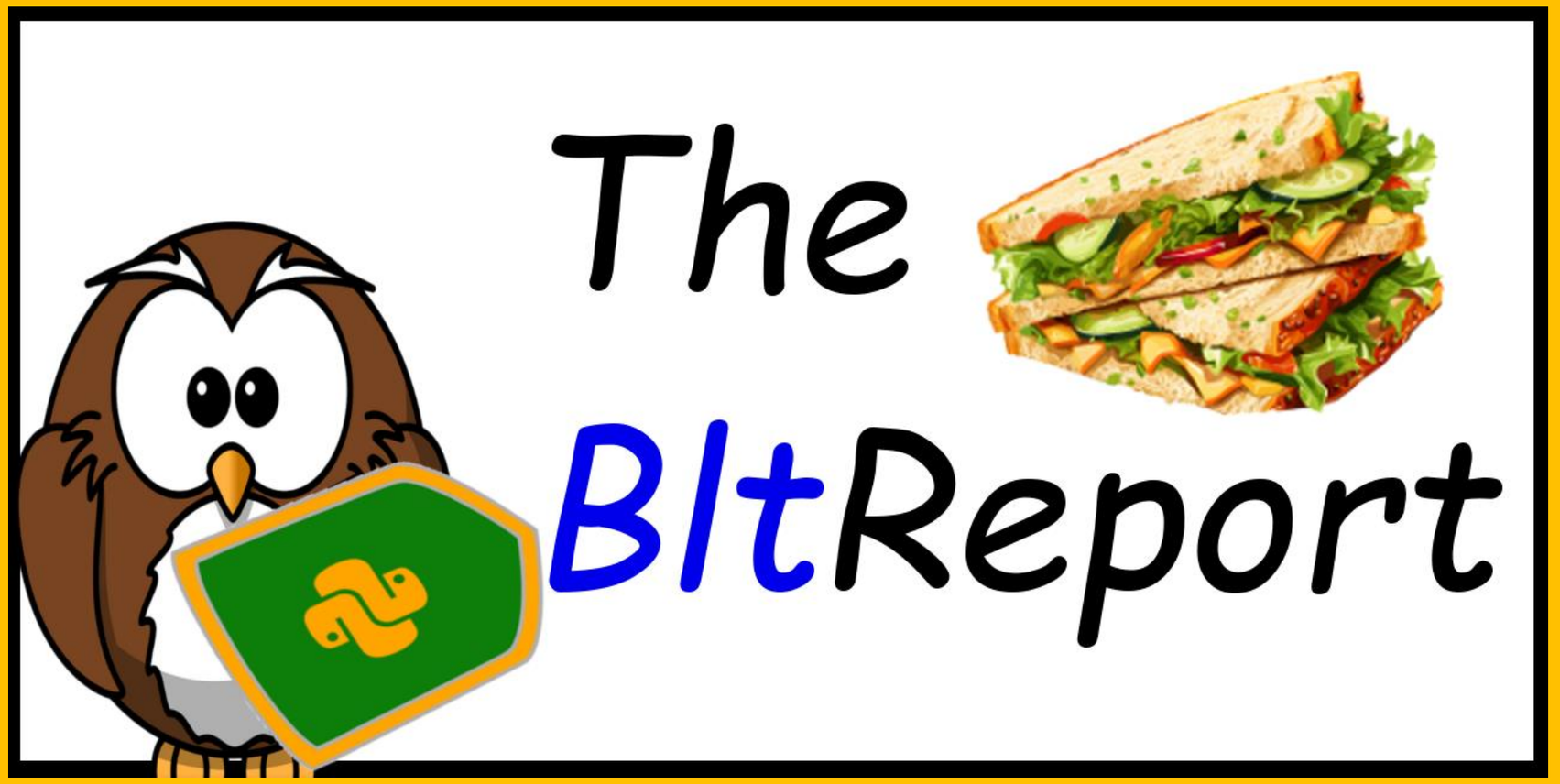
The Built-In Reports

By Randall Nagy

A.K.A: PyQuesting!



Video: BLT_00100



The ‘Upper-Cased’

Keywords:

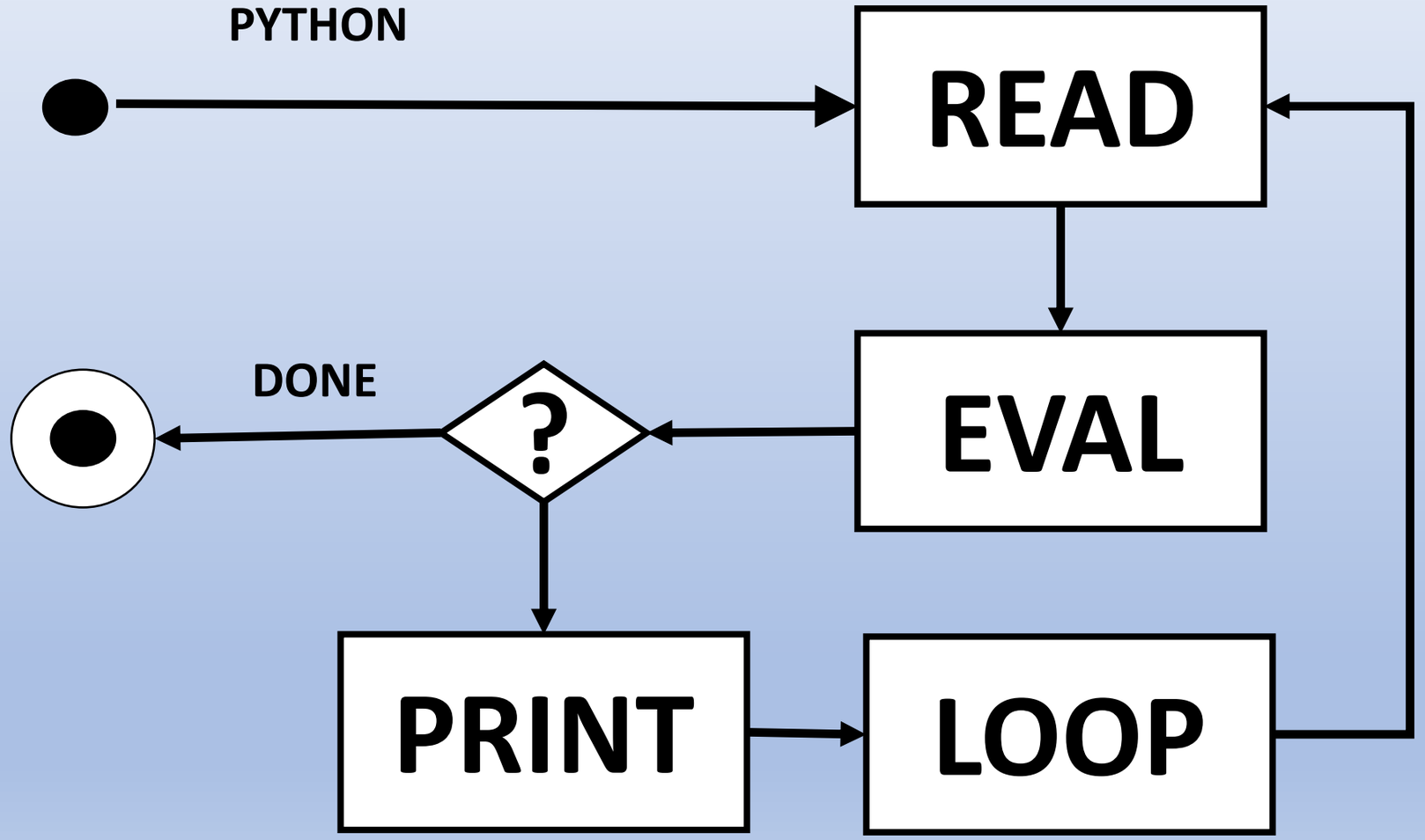
- True
- False
- None

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

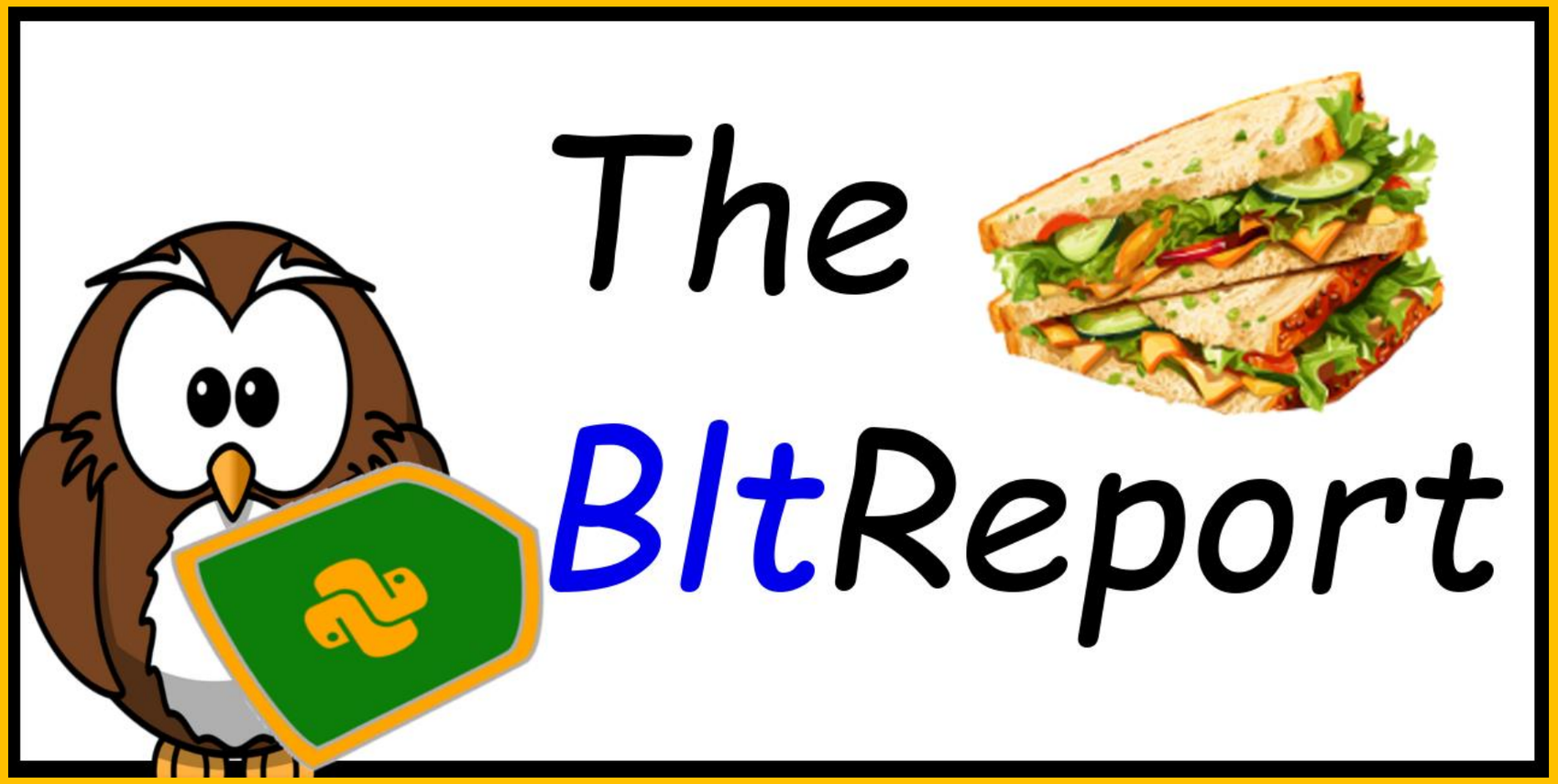
KA1060: Evaluations

Beginner

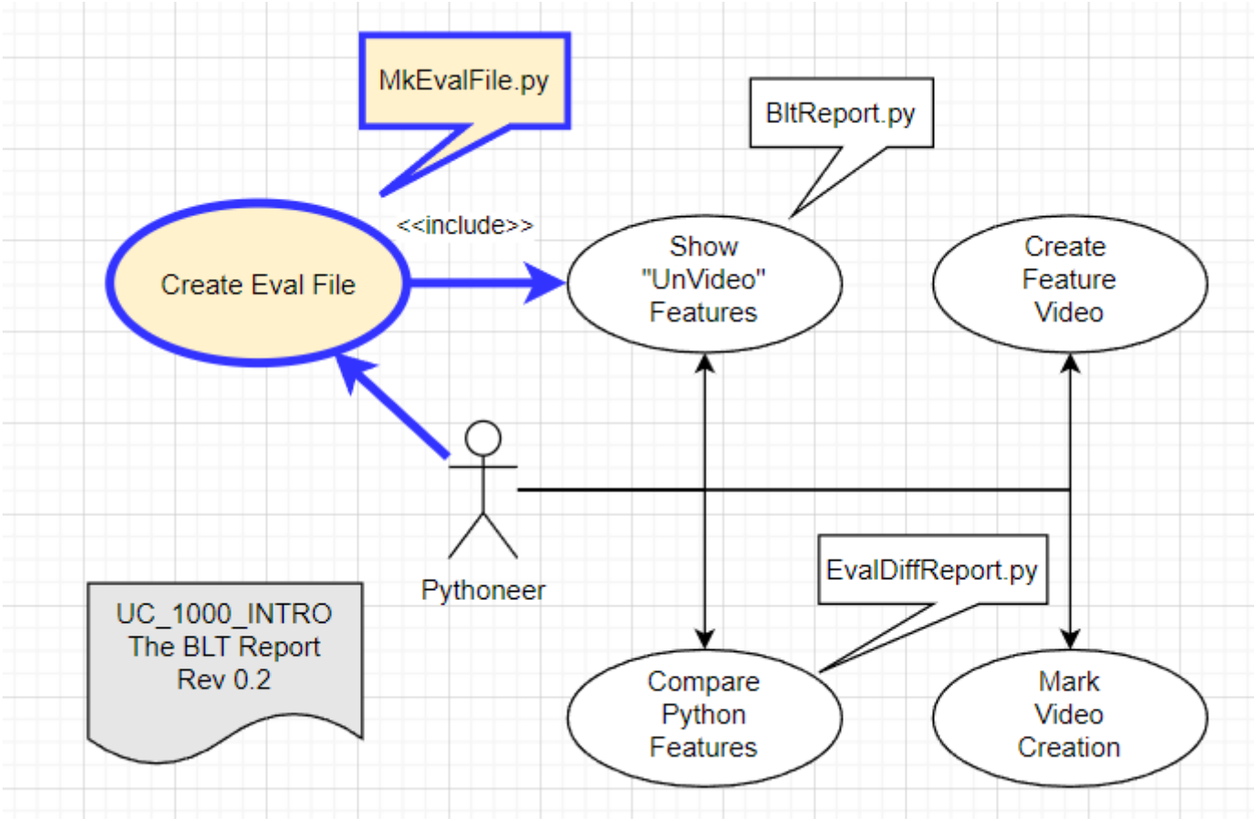
```
eval('bool(1)') will:
```

(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

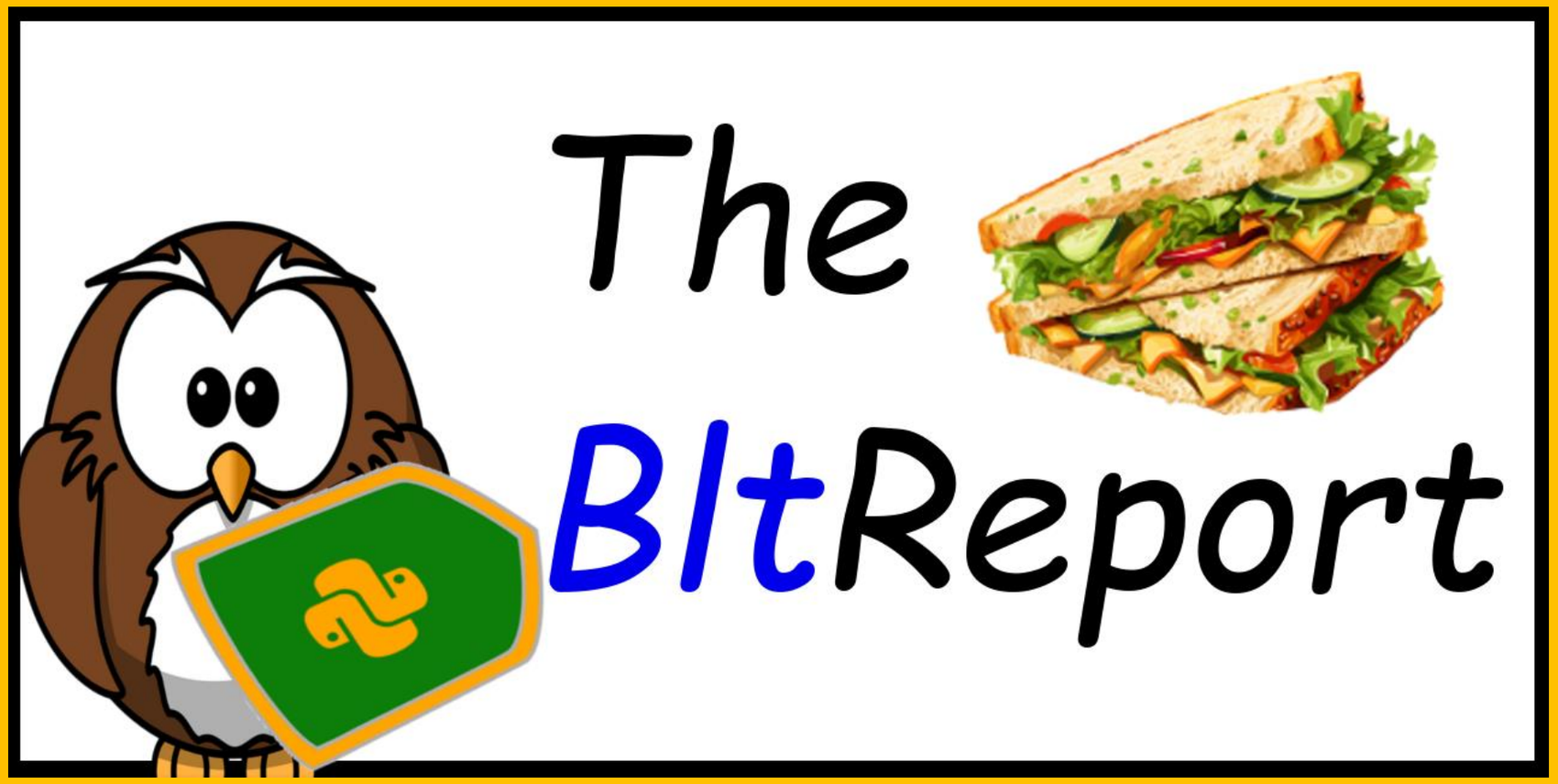
KA3036: Type Management

Advanced

Use type() to:

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

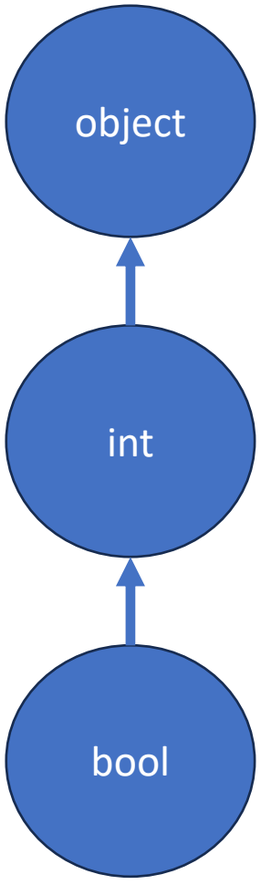
Video: BLT_00300



▶ 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(bool, int)
  - issubclass(int, bool)

Review: type() Initialization

- Case Study: BltTypeEx.py

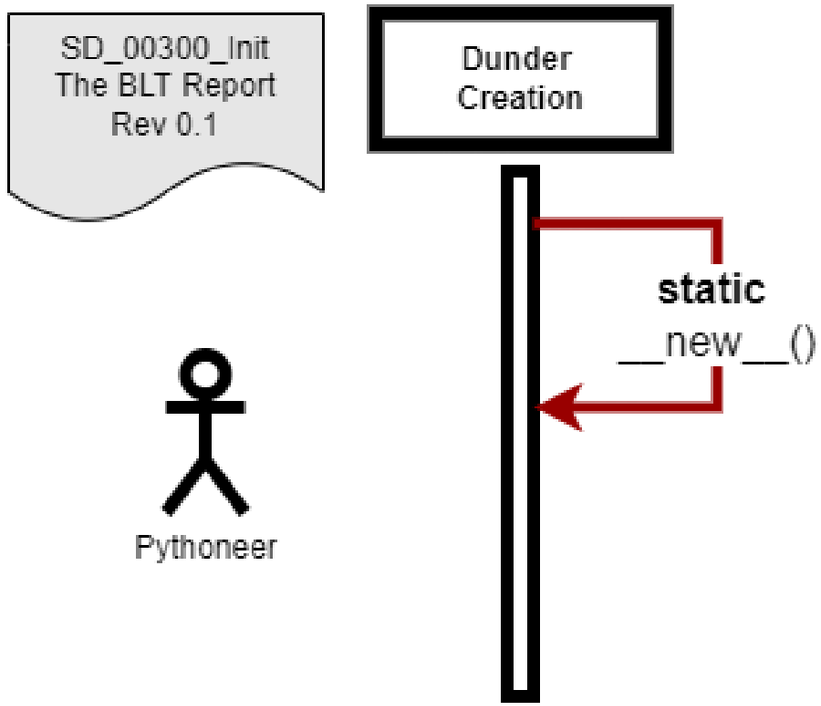
```
class zclass:
    def __init__(self, **kwargs):
        self.times = 1000
        print(f'Created zclass {kwargs}')

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

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

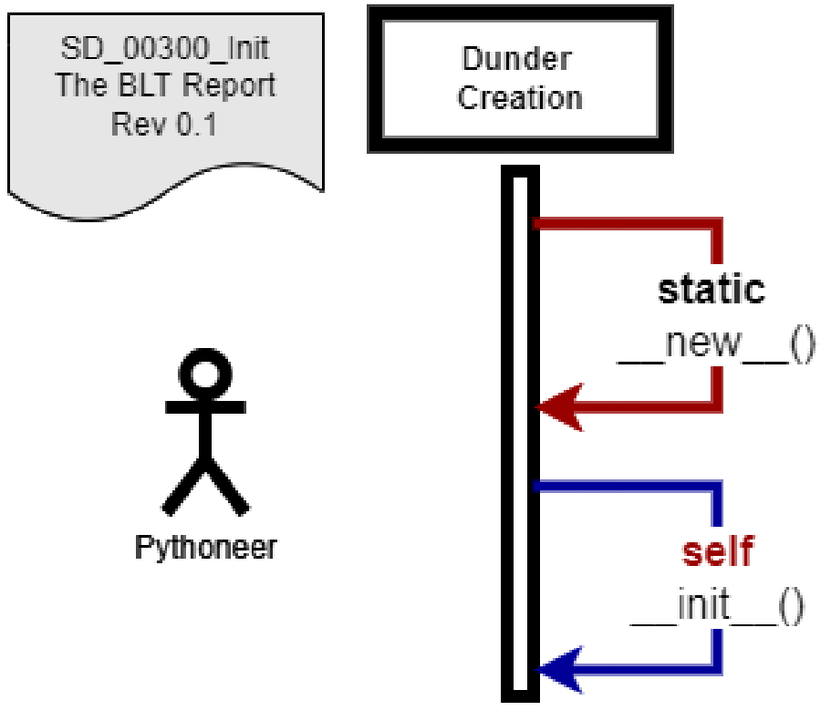
Review: Classic Initialization …

- 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

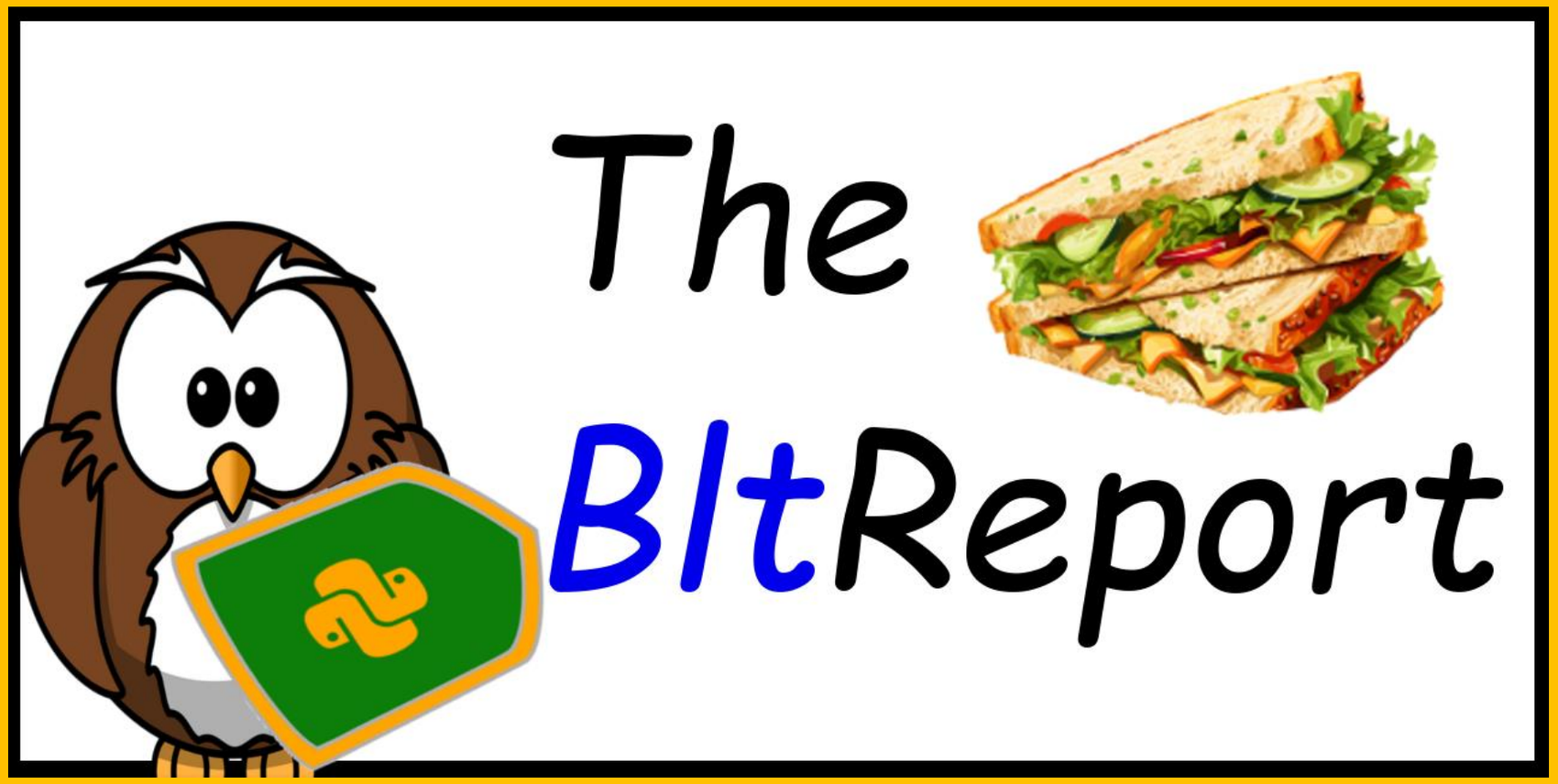
KA3038: Object Management

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

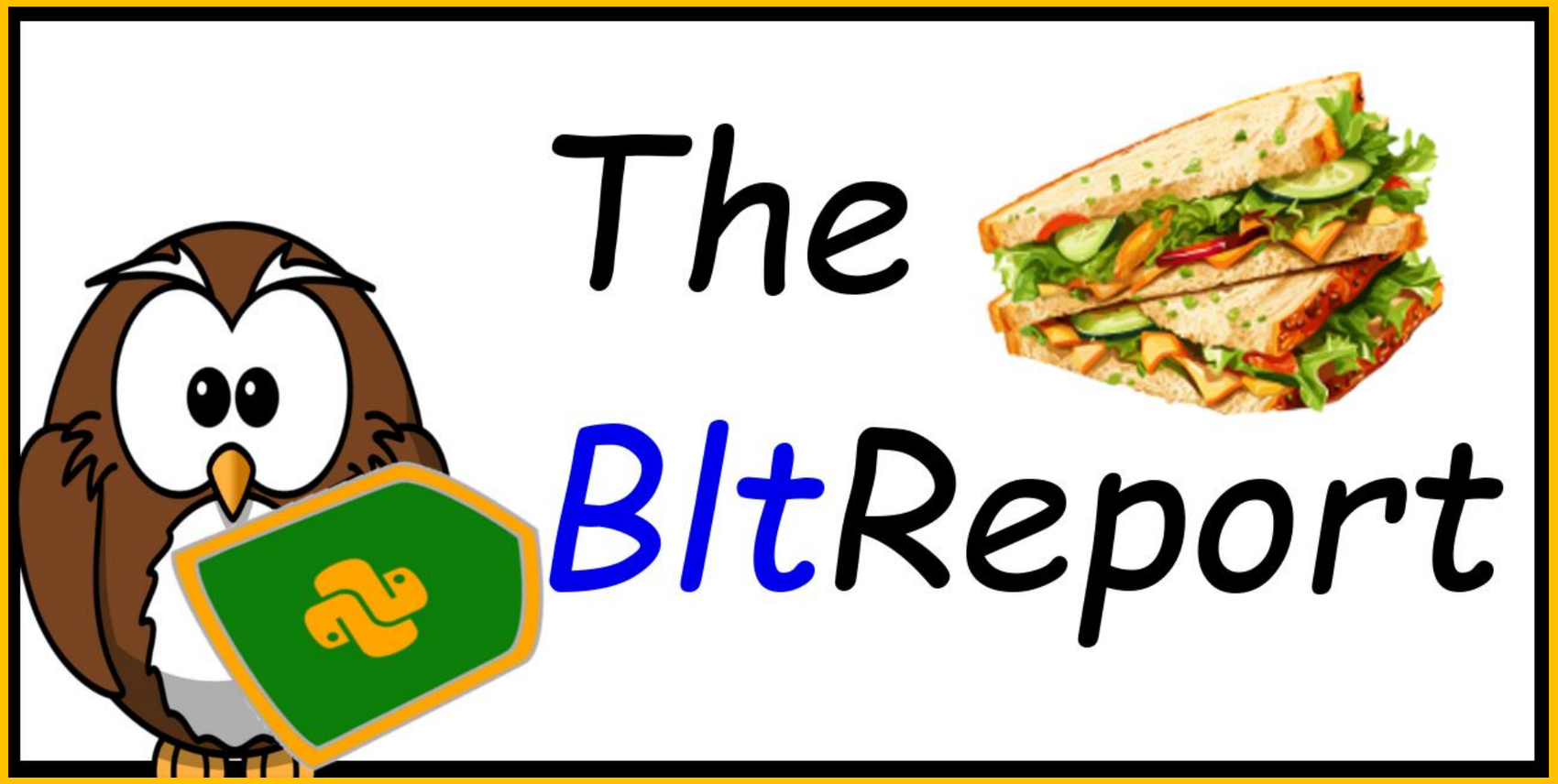
KA3041: Iteration Management

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

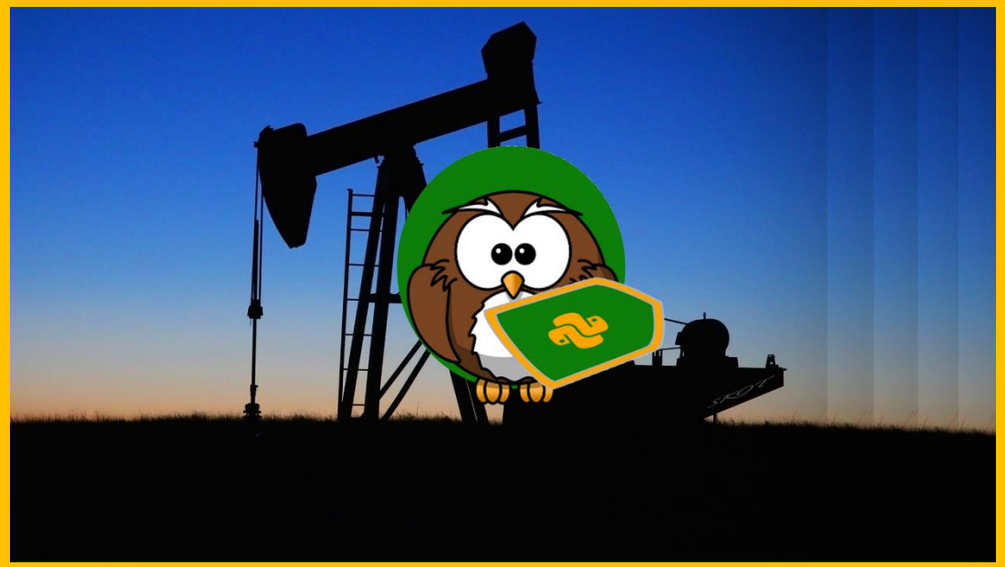
Intermediate

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

Modern Python

Happy PyQuesting!



(presentation end)