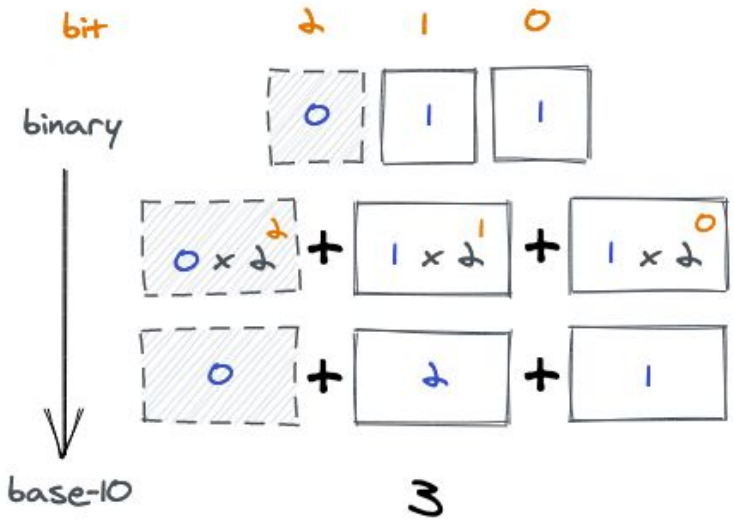
Python Curriculum

Part 02 - Logic Controls (2/2)

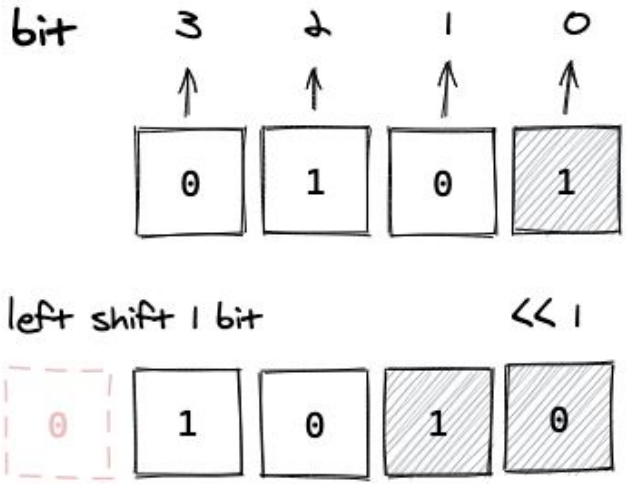
Bitwise Operations

```
>>> 0 | 1  # bitwise OR
1
>>> 0 & 1  # bitwise AND
0
>>> 0b010 | 0b001
3
>>> bin(3)
'0b11'
>>> int('0b11', 2)
3
```



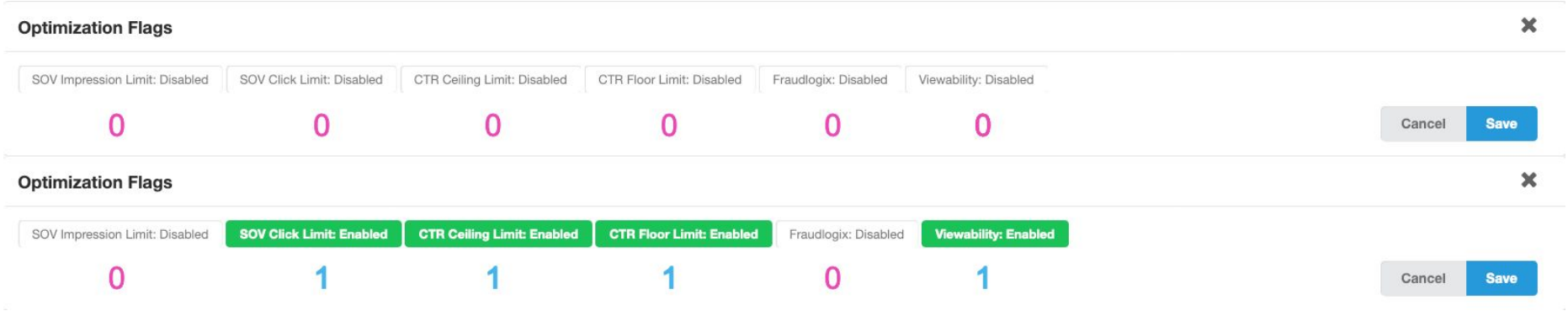
```
>>> bin(0b010 << 1)  # left shift and show binary string representation
'0b100'
>>> bin(0b010 >> 1)  # right shift and show binary string repr
'0b1'  # leading 0s are omitted
```

```
>>> x = 5
>>> x * 2 == x << 1
True
>>> x // 2 == x >> 1
True
```



```
>>> config = 0b1010_1010
>>> bin(config | 0b1111_0000)  # turn ON bits 7-4, leave bits 3-0 intact
'0b11111010'
>>> bin(config & 0b0000_1111)  # turn OFF bits 7-4, leave bits 3-0 intact
'0b1010'
>>> bin(config ^ 0b1111_1111)  # toggle all bits (using XOR, exclusive OR)
'0b1010101'  # leading 0 omitted, conceptually 0b0101_0101
>>> (config & 0b0100_1000) == 0b0100_1000  # query if bit 6 and 3 are both on
False
>>> bin(~0b01)  # negate
'-0b10'  # conceptually should just be 0b10
>>> bin(~0b01 & 0b11)  # force unsigned to be signed
'0b10'
```

Bitmasking




```
>>> SUN = 0
>>> MON = 1
>>> TUE = 2
>>> WED = 3
>>> THU = 4
>>> FRI = 5
>>> SAT = 6
>>> def turn_on_day(days, day_bit):
...     return days | (1 << day_bit)
...
>>> days = 0b0000000  # 7 days in bits, from right-to-left, SUN to SAT
>>> days = turn_on_day(days, SAT)  # turn on SAT
>>> bin(days)  # examine the binary string representation of resulting days
'0b1000000'
```

Error Controls and Controlled Errors

try/except

```
>>> 1 / 0
Traceback (most recent call last):
  ...
ZeroDivisionError: division by zero
```

```
>>> try:
...     1 / 0
... except:
...     pass
```

```
>>> a = 0
>>> b = '5'
>>> try:
...     b / a
... except ZeroDivisionError:
...     pass
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

```
>>> b = 'bob'
>>> a = '5'
>>> try:
...     c = b / a
... except ZeroDivisionError:
...     # handle zero division error when a == 0
...     c = a / b
... except TypeError:
...     # handle type error
...     try:
...         c = int(b) / int(a)
...     except ValueError:
...         print('we cannot perform {0} ({1}) / {2} ({3}'.format(
...             b, type(b), a, type(a)))
... except:
...     # theoretical last resort to catch any other errors
...     pass
...
we cannot perform bob (<class 'str'>) / 5 (<class 'str'>
```

Remember `hours_from()`?

```
'''module: utility.py'''
def hours_from(x, y):
    try:
        x = int(x)
        y = int(y)
    except ValueError:
        return None

    from_x = x + y  # unbound y hours from x
    from_x = str(from_x % 24)  # 24-hour capped hours from x, then cast to str
    z = from_x.zfill(2) + ':00'  # left-pad and format hours from x as HH:00
    return z  # return the value of z
```

```
>>> import utility
>>> utility.hours_from(16, 12345)  # utility module's hours_from() function
'01:00'
>>> utility.hours_from('16:00', 12345)
>>> # None, null, nil, nothing
```

Controlled Errors

```
'''module: utility.py'''
def hours_from(x, y):
    try:
        x = int(x)
        y = int(y)
    except ValueError:
        raise Exception('x and y need to be real numbers or base-10 number strings')

    from_x = x + y  # unbound y hours from x
    from_x = str(from_x % 24)  # 24-hour capped hours from x, then cast to str
    z = from_x.zfill(2) + ':00'  # left-pad and format hours from x as HH:00
    return z  # return the value of z
```

```
>>> from utility import hours_from  # cherry-pick hours_from() function from utility module
>>> hours_from('16:00', 123)
Traceback (most recent call last):
  ...
    x = int(x)
ValueError: invalid literal for int() with base 10: '16:00'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  ...
    raise Exception('x and y need to be real numbers or base-10 number strings')
Exception: x and y need to be real numbers or base-10 number strings
```

```
def hours_from(x, y):
    try:
        x = int(x)
        y = int(y)
    except ValueError:
        raise

    from_x = x + y  # unbound y hours from x
    from_x = str(from_x % 24)  # 24-hour capped hours from x, then cast to str
    z = from_x.zfill(2) + ':00'  # left-pad and format hours from x as HH:00
    return z  # return the value of z
```

```
def hours_from(x, y):
    x = int(x)
    y = int(y)

    from_x = x + y  # unbound y hours from x
    from_x = str(from_x % 24)  # 24-hour capped hours from x, then cast to str
    z = from_x.zfill(2) + ':00'  # left-pad and format hours from x as HH:00
    return z  # return the value of z
```

```
>>> from utility import hours_from
>>> hours_from('16:00', 12345)
Traceback (most recent call last):
  ...
    x = int(x)
ValueError: invalid literal for int() with base 10: '16:00'
```

Assertion

```
>>> from utility import hours_from
>>> assert hours_from(16, 12345) == '01:00'  # nothing, good
>>> assert hours_from(16, 12345) != '16:00'  # nothing, good
>>> assert hours_from(16, 12345) == '16:00'
Traceback (most recent call last):
  ...
AssertionError
```

if/else

vs

try/except

Questions?