Meeting 3 Live

December 15, 2025

[1]:
```
my_tuple = (0, 1, 2)
```

[2]:
```
my_tuple[0]
```

[2]:
```
0
```

[3]:
```
my_tuple[0] = 3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 my_tuple[0] = 3

TypeError: 'tuple' object does not support item assignment
```

[4]:
```
my_tuple_of_lists = (['Ben', 'S.'], ['Simon', 'E.'])
```

[5]:
```
for entry in my_tuple_of_lists:
    print(entry)
```

```
['Ben', 'S.']
['Simon', 'E.']
```

[7]:
```
for first, last in my_tuple_of_lists:
    print(first, last)
```

```
Ben S.
Simon E.
```

[8]:
```
enumerate?
```

```
Init signature: enumerate(iterable, start=0)
Docstring:
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.

enumerate is useful for obtaining an indexed list:
    (0, seq[0]), (1, seq[1]), (2, seq[2]), …
Type:           type
Subclasses:
```

[9]:
```
for i, entry in enumerate(my_tuple_of_lists):
    print(i, entry)
```

```
0 ['Ben', 'S.']
1 ['Simon', 'E.']
```

[10]:
```
for i, first, last in enumerate(my_tuple_of_lists):
    print(i, entry)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[10], line 1
----> 1 for i, first, last in enumerate(my_tuple_of_lists):
      2     print(i, entry)

ValueError: not enough values to unpack (expected 3, got 2)
```

[11]:
```
for i, (first, last) in enumerate(my_tuple_of_lists):
    print(i, first, last)
```

```
0 Ben S.
1 Simon E.
```

[16]:
```
staff = ['Ben', 'Simo']
```

[17]:
```
for entry in staff:
    if entry == 'Simon':
        print("One Simon found")
        break
else:
    print("We need one Simon")
```

```
We need one Simon
```

[18]:
```
def get_square_numbers(n: int) -> list:
    result = []
    for i in range(1, n + 1):
        result.append(i ** 2)
```

```
    return result

get_square_numbers(10)
```

[18]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

[19]:
```
def get_square_numbers_compr(n: int) -> list:
    return [i ** 2 for i in range(1, n + 1)]
```

[20]:
```
get_square_numbers_compr(10)
```

[20]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

[21]:
```
def get_square_numbers_compr(n: int) -> list:
    result = [i ** 2 for i in range(1, n + 1)]
    return result

get_square_numbers_compr(10)
```

[21]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

[22]:
```
def get_even_square_numbers(n: int) -> list:
    result = []
    for i in range(1, n + 1):
        if i % 2 == 0:
            result.append(i ** 2)
    return result

get_even_square_numbers(10)
```

[22]:
```
[4, 16, 36, 64, 100]
```

[23]:
```
def get_even_square_numbers_compr(n: int) -> list:
    result = [i ** 2 for i in range(1, n + 1) if i % 2 == 0]
    return result
get_even_square_numbers_compr(10)
```

[23]:
```
[4, 16, 36, 64, 100]
```

[25]:
```
def get_divisible_by_two_and_three_squares(n: int) -> list:
    result = []
    for i in range(1, n + 1):
        if i % 2 == 0 and i % 3 == 0:
            result.append(i ** 2)
    return result

get_divisible_by_two_and_three_squares(10)
```

[25]: [36]

[26]:
```
def get_divisible_by_two_and_three_squares_compr(n: int) -> list:
    result = [i ** 2 for i in range(1, n + 1) if i % 2 == 0 and i % 3 == 0]
    return result
get_divisible_by_two_and_three_squares_compr(10)
```

[26]: [36]

[27]:
```
with open("test.txt", "r") as fh:
    print(fh.readlines())
```

```
['Testing 1234\n', 'Hello World\n', 'Hallo Welt']
```

[28]:
```
with open("test.txt", "r") as fh:
    print(fh.read())
```

```
Testing 1234
Hello World
Hallo Welt
```

[30]:
```
fh = open("test2.txt", "w")
```

[31]:
```
import time
fh.write("foo")
time.sleep(5)
```

[32]:
```
fh.close()
```

[33]:
```
with open("test2.txt", "w") as fh:
    fh.write("foobar")
```

[34]:
```
with open("test2.txt", "a") as fh:
    fh.write("\nHello World")
```

[35]:
```
"foo".encode()
```

[35]: b'foo'

[36]:
```
b'foo'.decode()
```

[36]: 'foo'

[37]:
```
with open("test2.txt", "rb") as fh:
    print(fh.read())
```

```
b'foobar\nHello World'
```

[38]:
```
my_str = "foo"
```

[39]:
```
my_str[0] = "F"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[39], line 1
----> 1 my_str[0] = "F"

TypeError: 'str' object does not support item assignment
```

[40]:
```
my_tuple[0] = "foo"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[40], line 1
----> 1 my_tuple[0] = "foo"

TypeError: 'tuple' object does not support item assignment
```

[41]:
```
my_str[3]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[41], line 1
----> 1 my_str[3]

IndexError: string index out of range
```

[42]:
```
my_str[3]
print("Hello World")
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[42], line 1
----> 1 my_str[3]
      2 print("Hello World")

IndexError: string index out of range
```

[43]:
```
try:
    my_str[3]
except IndexError:
    print("IndexError occurred")
```

```
print("Hello World")
```

```
IndexError occurred
Hello World
```

[44]:
```
try:
    my_str[0] = "F"
except IndexError:
    print("IndexError occurred")
print("Hello World")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 2
      1 try:
----> 2     my_str[0] = "F"
      3 except IndexError:
      4     print("IndexError occurred")

TypeError: 'str' object does not support item assignment
```

[46]:
```
try:
    my_str[0] = "F"
    print("After broken code")
except IndexError:
    print("IndexError occurred")
except TypeError:
    print("TypeError occurred")
print("Hello World")
```

```
TypeError occurred
Hello World
```

[47]:
```
1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[47], line 1
----> 1 1/0

ZeroDivisionError: division by zero
```

[48]:
```
try:
    1/0
except IndexError:
    print("IndexError occurred")
```

```
except TypeError:
    print("TypeError occurred")
print("Hello World")
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[48], line 2
      1 try:
----> 2     1/0
      3 except IndexError:
      4     print("IndexError occurred")

ZeroDivisionError: division by zero
```

[49]:
```
try:
    1/0
except IndexError:
    print("IndexError occurred")
except TypeError:
    print("TypeError occurred")
except Exception:
    print("Some exception occurred")
print("Hello World")
```

```
Some exception occurred
Hello World
```

[50]:
```
try:
    1/0
except IndexError:
    print("IndexError occurred")
except TypeError:
    print("TypeError occurred")
except Exception as e:
    print(f"{e.__class__} occurred")
print("Hello World")
```

```
<class 'ZeroDivisionError'> occurred
Hello World
```

[51]:
```
def is_prime(n: int) -> bool:
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

[52]:
```
is_prime(7)
```

[52]:
```
True
```

[53]:
```
is_prime(8)
```

[53]:
```
False
```

[54]:
```
is_prime(1)
```

[54]:
```
True
```

[55]:
```
def is_prime(n: int) -> bool:
    if n == 1:
        raise ValueError("1 is not a prime!!!1einself")
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

[56]:
```
is_prime(1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[56], line 1
----> 1 is_prime(1)

Cell In[55], line 3, in is_prime(n)
      1 def is_prime(n: int) -> bool:
      2     if n == 1:
----> 3         raise ValueError("1 is not a prime!!!1einself")
      4     for i in range(2, n):
      5         if n % i == 0:

ValueError: 1 is not a prime!!!1einself
```

[57]:
```
is_prime(0.1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[57], line 1
----> 1 is_prime(0.1)

Cell In[55], line 4, in is_prime(n)
      2 if n == 1:
      3     raise ValueError("1 is not a prime!!!1einself")
```

```
----> 4 for i in range(2, n):
      5     if n % i == 0:
      6         return False

TypeError: 'float' object cannot be interpreted as an integer
```

[58]:
```
def is_prime(n: int) -> bool:
    if not isinstance(n, int):
        raise TypeError("n must be an integer!")
    if n == 1:
        raise ValueError("1 is not a prime!!!1einself")
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

[59]:
```
is_prime(0.1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[59], line 1
----> 1 is_prime(0.1)

Cell In[58], line 3, in is_prime(n)
      1 def is_prime(n: int) -> bool:
      2     if not isinstance(n, int):
----> 3         raise TypeError("n must be an integer!")
      4     if n == 1:
      5         raise ValueError("1 is not a prime!!!1einself")

TypeError: n must be an integer!
```

[60]:
```
import math
```

[61]:
```
math.factorial(5)
```

[61]: 120

[62]:
```
math.gcd(5, 17)
```

[62]: 1

[63]:
```
math.pi
```

[63]: 3.141592653589793

[64]:
```
dir(math)
```

[64]:
```
['__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'comb',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'dist',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
 'isclose',
 'isfinite',
 'isinf',
 'isnan',
 'isqrt',
 'lcm',
 'ldexp',
 'lgamma',
 'log',
 'log10',
```

```
 'log1p',
 'log2',
 'modf',
 'nan',
 'nextafter',
 'perm',
 'pi',
 'pow',
 'prod',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
 'tau',
 'trunc',
 'ulp']
```

[65]:
```
import string
string.ascii_letters
```

[65]: 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

[66]:
```
string.ascii_lowercase
```

[66]: 'abcdefghijklmnopqrstuvwxyz'

[67]:
```
string.ascii_uppercase
```

[67]: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

[68]:
```
string.hexdigits
```

[68]: '0123456789abcdefABCDEF'

[69]:
```
import hashlib
```

[70]:
```
hashlib.md5("foo".encode())
```

[70]: <md5 _hashlib.HASH object @ 0x7fbee6ae8c70>

[71]:
```
hashlib.md5("foo".encode()).hexdigest()
```

[71]: 'acbd18db4cc2f85cedef654fccc4a4d8'

[72]:
```
hashlib.sha1("foo".encode()).hexdigest()
```

[72]:
```
'0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33'
```

[73]:
```
hashlib.sha256("foo".encode()).hexdigest()
```

[73]:
```
'2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae'
```

[74]:
```
import json
```

[75]:
```
json.dumps([1, 2])
```

[75]:
```
'[1, 2]'
```

[76]:
```
json.loads('[1, 2]')
```

[76]:
```
[1, 2]
```

[78]:
```
json.dumps({'Simon': 'happy', 'Ben': 'n *= i'})
```

[78]:
```
'{"Simon": "happy", "Ben": "n *= i"}'
```

[ ]:
```

```