Break

Thomas Schwarz, SJ

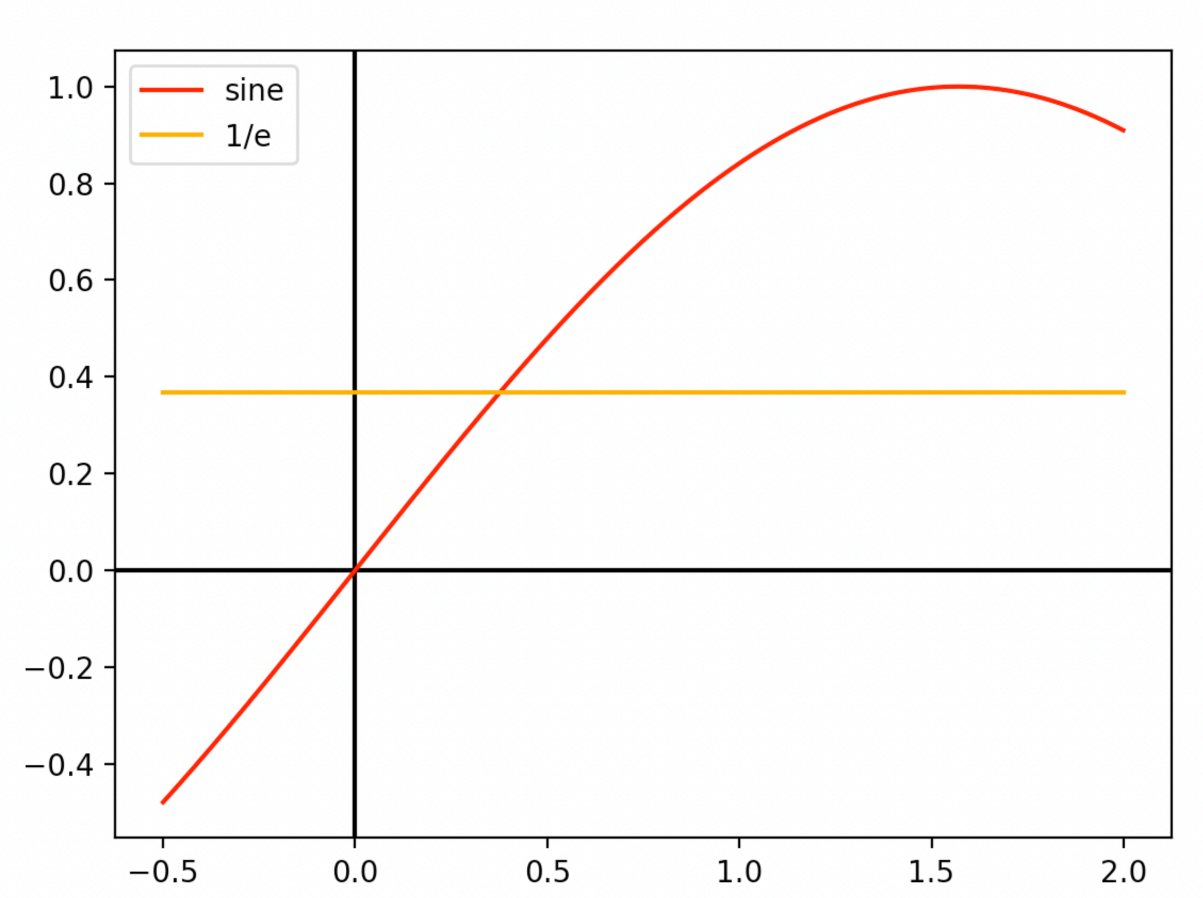
Break Statement

- Recall: The `break` statement jumps out of the closest enclosing loop
  - Obviously, used conditionally

```
if condition:
    break
```

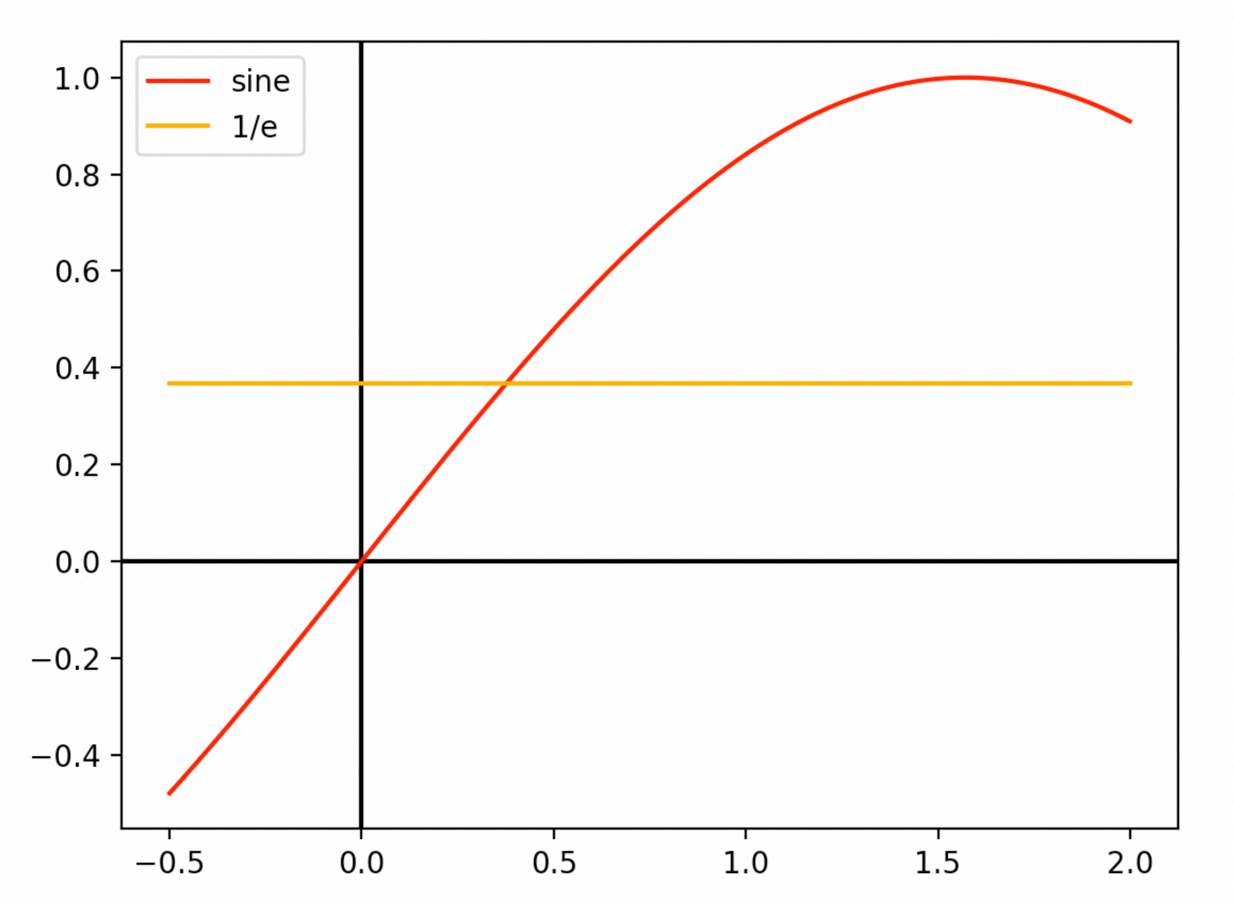
Break Statement

- Paradigm: The while-true loop
  - Example:
    - Solving $\sin(x) = \frac{1}{e}$
- To calculate:
  - import math
  - use math.sin and math.e



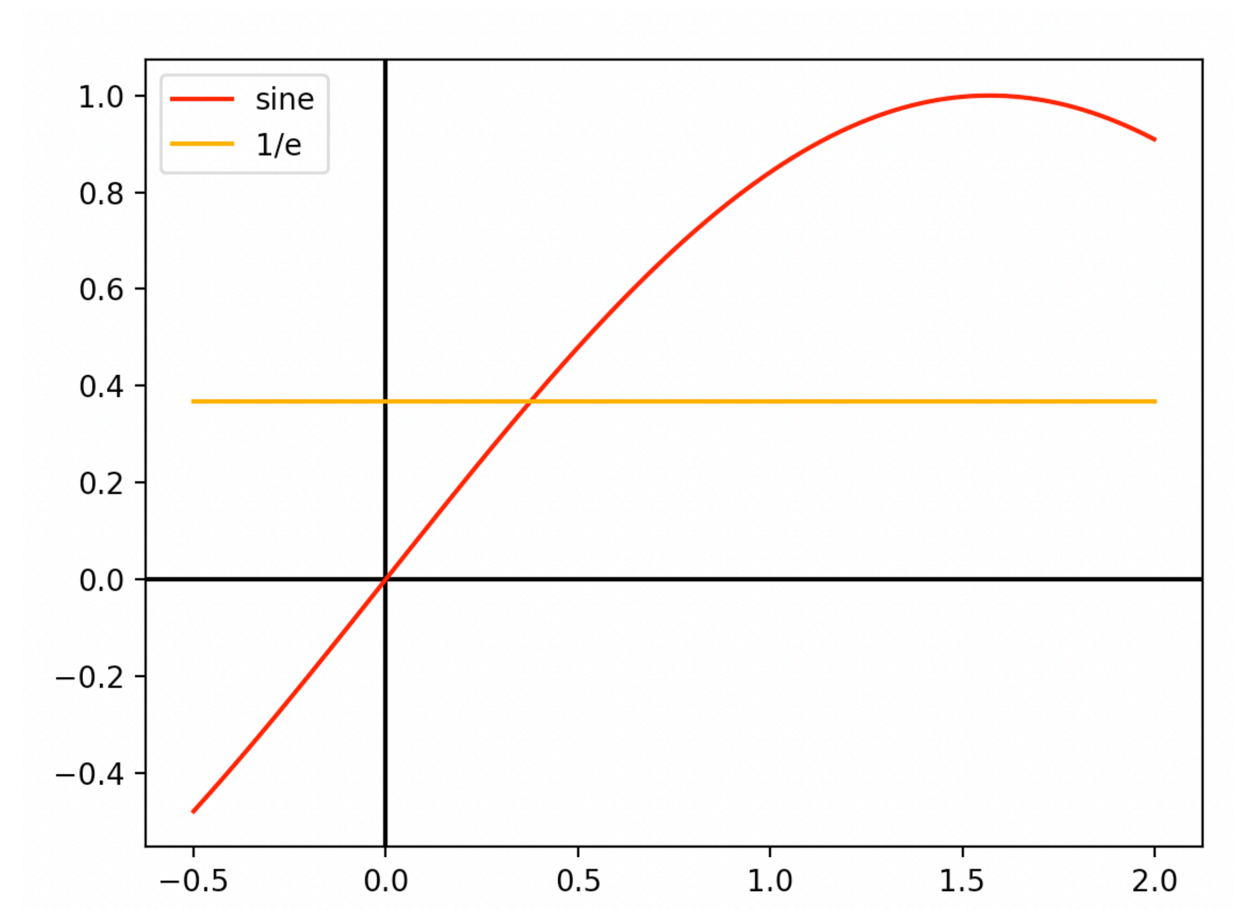
Break Statement

- Strategy:
  - As we can see, the solution lies between 0 and 1
  - We try out 0.5
  - Since `math.sin(0.5) > 1/math.e`
    - Solution between 0 and 0.5



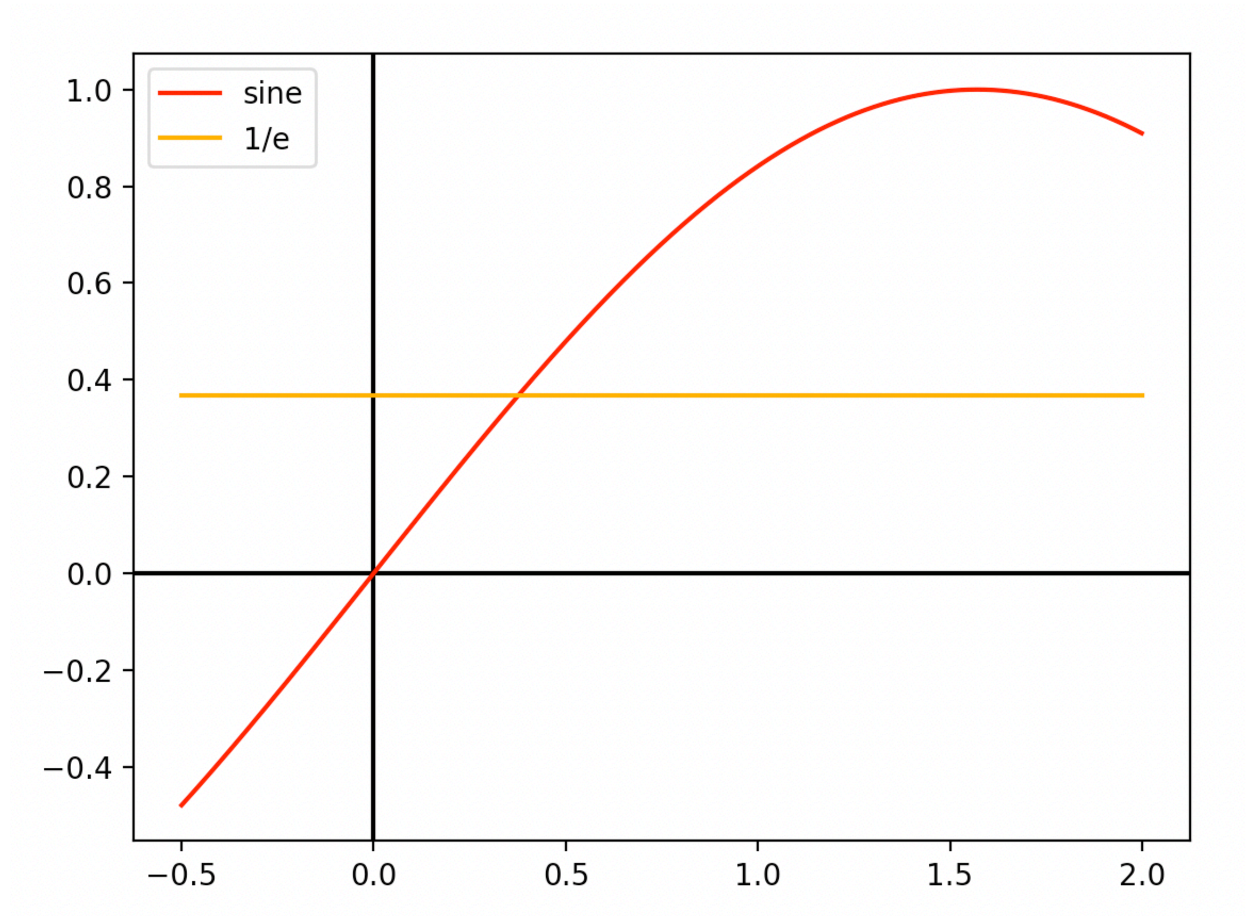
Break Statement

- Strategy:
  - We try out 0.25
  - Since `math.sin(0.25) < 1/math.e`
    - Solution between 0.25 and 0.5



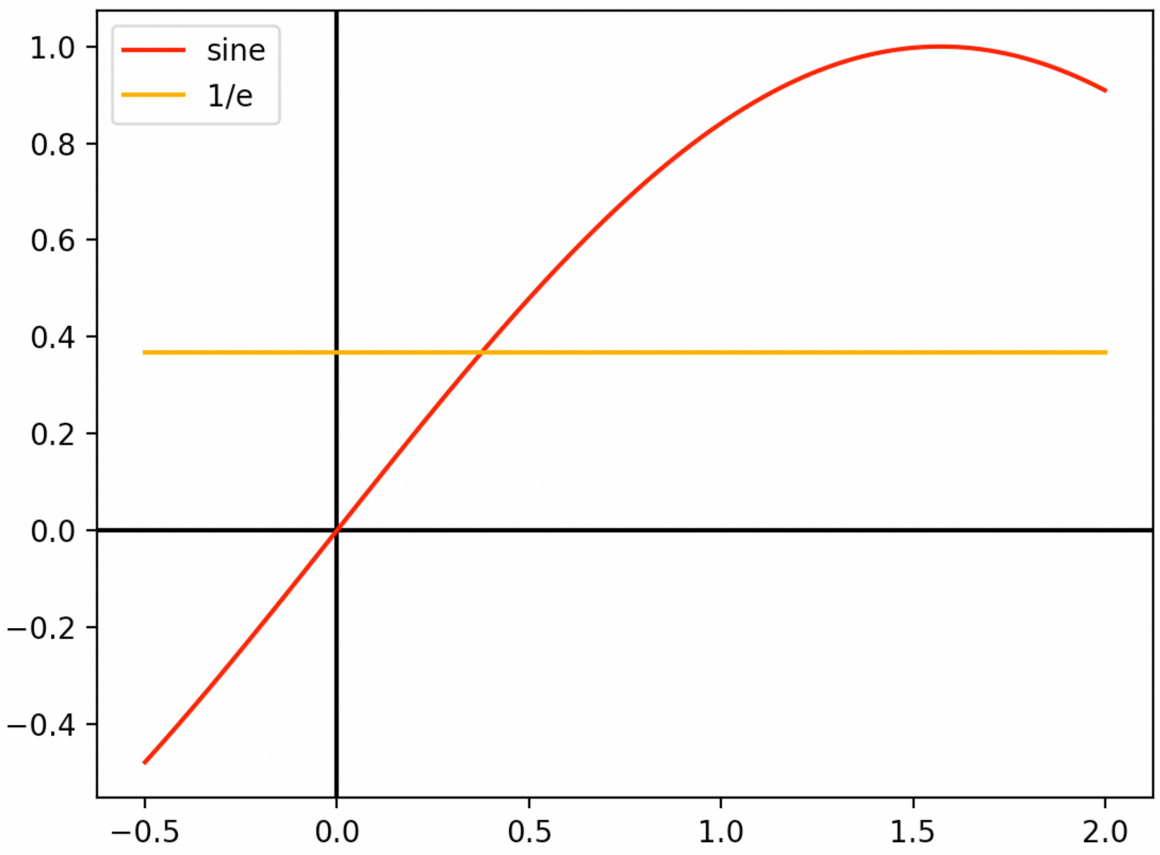
Break Statement

- Strategy:
  - We try out 0.375
  - Since `math.sin(0.375) < 1/math.e`
    - Solution between 0.375 and 0.5



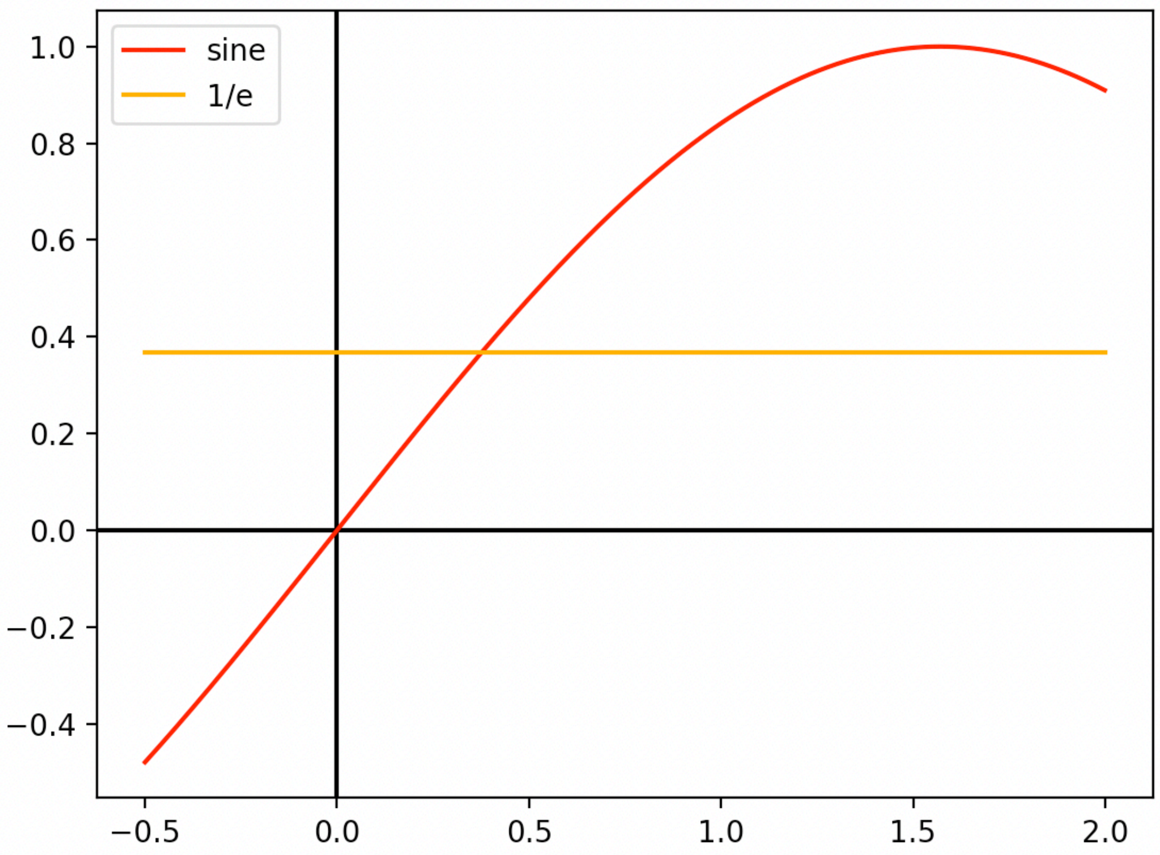
Break Statement

- Strategy:
  - We try out (0.375+0.5)/2 = 0.4375
  - Since `math.sin(0.4375) > 1/math.e`
    - Solution between 0.375 and 0.4375



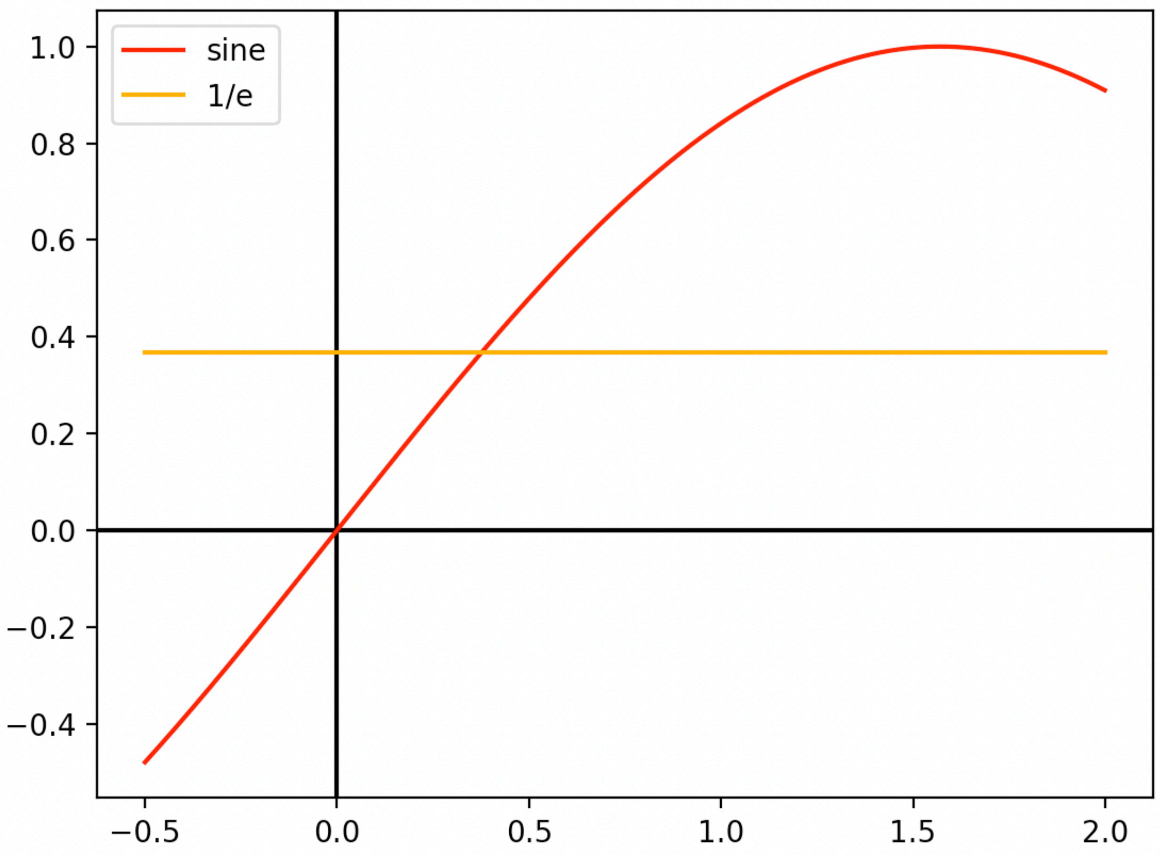
Break Statement

- Strategy:
  - Starting with an interval [a,b]
  - We form the mean $\frac{a+b}{2}$
  - We evaluate whether $\sin(\frac{a+b}{2}) > \frac{1}{e}$
  - and update the interval accordingly



Break Statement

- Strategy:
  - We stop when the size of the interval is small
  - E.g. less than 1/1000



Break Statement

- Code:

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

Break Statement

- Code:

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

import math so that we can use sin and e

Break Statement

- Code:

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

Initialize the interval boundaries

Break Statement

- Code:

An “infinite” loop

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

Break Statement

- Code:

Calculate the midpoint

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

Break Statement

- Code:

If the value is smaller, then the searched for value is to the right

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

Break Statement

- Code:

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

But we only need to change a, b can stand as it is

Break Statement

- Code:

Otherwise, midpoint is to the right, so we need to change b

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

Break Statement

- Code:

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

If the interval is small, we have found the value

Break Statement

- Code:

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

To look at the changes is instructive
We can comment this out

Break Statement

- Code:

```
import math
a = 0
b = 1

while True:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [midpoint,b ]
        a = midpoint
    else:
        b = midpoint
    if b-a < 1/100000:
        break
    print(a,b)
print(a, b)
```

We need to announce our result

Break Statement

```
0 0.5
0.25 0.5
0.375 0.5
0.375 0.4375
0.375 0.40625
0.375 0.390625
0.375 0.3828125
0.375 0.37890625
0.375 0.376953125
0.3759765625 0.376953125
0.37646484375 0.376953125
0.376708984375 0.376953125
0.376708984375 0.3768310546875
0.376708984375 0.37677001953125
0.376708984375 0.376739501953125
0.3767242431640625 0.376739501953125
0.3767242431640625 0.37673187255859375
0.3767242431640625 0.3767280578613281
0.3767261505126953 0.3767280578613281
0.3767271041870117 0.3767280578613281
```

Difference set to < 1/100000 (six digits accuracy)

Break Statement

- Now that we are done, we can consider replacing the True with a condition
- As there is only one condition, the choice is easy

```
import math
a = 0
b = 1

while b-a>1/1000000:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [ (a+b)/2,b ]
        a = midpoint
    else:
        b = midpoint
print(a, b)
```

Break Statement

- The use of 1/100000 is ugly
  - We replace it with a constant

```
import math
a = 0
b = 1
PRECISION = 1/1000000
while b-a>PRECISION:
    midpoint = (a+b)/2
    if math.sin(midpoint) < 1/math.e: #new interval [ (a+b)/2,b ]
        a = midpoint
    else:
        b = midpoint
print(a, b)
```