Continue Statement

Thomas Schwarz, SJ

Continue

- A continue statement breaks out of the current execution of the loop
  - But goes to the next instance of the loop

Continue

- Example 1:

```
for i in range(1, 100):
    if i % 3 != 0:
        continue
    print(i, i**2, i**3)
```

Continue

- Example 1:

```
for i in range(1, 100):
    if i % 3 != 0:
        continue
    print(i, i**2, i**3)
```

First: i=1

Continue

- Example 1:

```
for i in range(1, 100):
    if i % 3 != 0:
        continue
    print(i, i**2, i**3)
```

If is triggered

Continue

- Example 1:

```
for i in range(1, 100):
    if i % 3 != 0:
        continue
    print(i, i**2, i**3)
```

go back to the beginning of the loop

Continue

- Example 1:

```
for i in range(1, 100):
    if i % 3 != 0:
        continue
    print(i, i**2, i**3)
```

Second: i=2

Continue

- Example 1:

```
for i in range(1, 100):
    if i % 3 != 0:
        continue
    print(i, i**2, i**3)
```

We also jump back to the beginning of the loop

Continue

- Example 1:

```
for i in range(1, 100):
    if i % 3 != 0:
        continue
    print(i, i**2, i**3)
```

Next i is 3

Continue

- Example 1:

```
for i in range(1, 100):
    if i % 3 != 0:
        continue
    print(i, i**2, i**3)
```

We skip the continue

Continue

- Example 1:

```
for i in range(1, 100):
    if i % 3 != 0:
        continue
    print(i, i**2, i**3)
```

and print 3, 9, 27

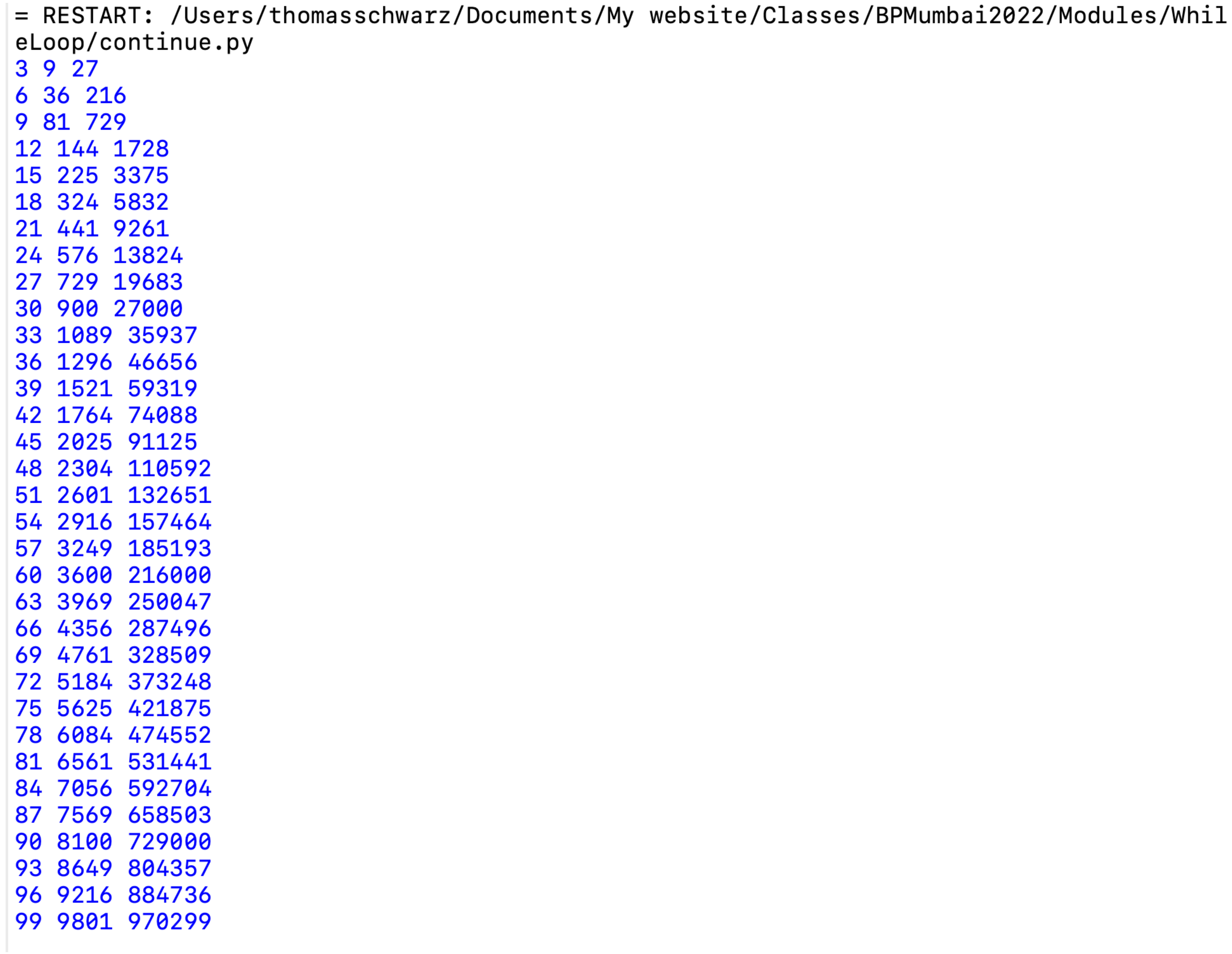
Continue

- Example 1:

```
for i in range(1, 100):
    if i % 3 != 0:
        continue
    print(i, i**2, i**3)
```

Then we go back and set i to 4

Continue



```
= RESTART: /Users/thomasschwarz/Documents/My website/Classes/BPMumbai2022/Modules/Whil
eLoop/continue.py
3 9 27
6 36 216
9 81 729
12 144 1728
15 225 3375
18 324 5832
21 441 9261
24 576 13824
27 729 19683
30 900 27000
33 1089 35937
36 1296 46656
39 1521 59319
42 1764 74088
45 2025 91125
48 2304 110592
51 2601 132651
54 2916 157464
57 3249 185193
60 3600 216000
63 3969 250047
66 4356 287496
69 4761 328509
72 5184 373248
75 5625 421875
78 6084 474552
81 6561 531441
84 7056 592704
87 7569 658503
90 8100 729000
93 8649 804357
96 9216 884736
99 9801 970299
```

Continue

- Continue and while can lend itself to an unintended infinite loop
    - Example 2:

```
i=1
while(i<100):
    if i % 3 == 0:
        continue
    print(i, i**2, i**3)
    i += 1
```

Continue

- Example 2:

```
i=1
while(i<100):
    if i % 3 == 0:
        continue
    print(i, i**2, i**3)
    i += 1
```

Intention: i=1, 2, 3, 4, 5, …
but skip over multiples of 3

Continue

- Example 2:

```
i=1
while(i<100):
    if i % 3 == 0:
        continue
    print(i, i**2, i**3)
    i += 1
```

We have not forgotten to increment, but if we execute the continue, we do not increment

Continue

- Example 2:

```
i=1
while(i<100):
    if i % 3 == 0:
        continue
    print(i, i**2, i**3)
    i += 1
```

If i = 3:
Execute continue, jump back to the loop, condition is true, enter the loop, execute continue, jump back to the loop, condition is true, enter the loop, execute continue, jump back to the loop

Continue

- Use case:
  - Searching for examples
  - Can abort a loop early

Amicable Numbers

- A number is called *perfect* if it equals the sum of its proper divisors
  - This includes 1 but excludes the number itself
  - For instance: 6 = 1+2+3
- As a “finger exercise”, we find the first 4 perfect numbers
  - After this, our search will take a long time
    - the fifth one is 33550336
- Then we go on to amicable numbers

Perfect Numbers

- An efficient algorithm uses number theory, so do not take this as the latest in the art
    - We are still learning
- When we program in Python, we can work incrementally
    - But we still need to have a plan before we start programming
        - For a given number:
            - Try out all numbers smaller and add them if they divide the number

Perfect Numbers

- First, here is how we can calculate the sum of divisors:

```
number = 267
sum_of_divisors = 0
for divisor in range(1,267):
    if number%divisor == 0:
        sum_of_divisors += divisor
print(number, sum_of_divisors)
```

Perfect Numbers

```
number = 267
sum_of_divisors = 0
for divisor in range(1,267):
    if number%divisor == 0:
        sum_of_divisors += divisor
print(number, sum_of_divisors)
```

An accumulator

Perfect Numbers

```
number = 267
sum_of_divisors = 0
for divisor in range(1,267):
    if number%divisor == 0:
        sum_of_divisors += divisor
print(number, sum_of_divisors)
```

Run through 1, ..., 266

Perfect Numbers

```
number = 267
sum_of_divisors = 0
for divisor in range(1,267):
    if number%divisor == 0:
        sum_of_divisors += divisor
print(number, sum_of_divisors)
```

Use floor division to determine whether divisor indeed divides number

Perfect Numbers

```
number = 267
sum_of_divisors = 0
for divisor in range(1,267):
    if number%divisor == 0:
        sum_of_divisors += divisor
print(number, sum_of_divisors)
```

Add divisors to sum_of_divisors

Perfect Numbers

- There is a small improvement, since we know that 1 divides the number

```
number = 267
sum_of_divisors = 1
for divisor in range(2,267):
    if number%divisor == 0:
        sum_of_divisors += divisor
print(number, sum_of_divisors)
```

Perfect Numbers

- We now can determine perfect numbers until 10,000

```
for number in range(2,10000):
    sum_of_divisors = 1
    for divisor in range(2,number):
        if number%divisor == 0:
            sum_of_divisors += divisor
    if number == sum_of_divisors:
        print(number, 'is perfect')
```

Perfect Numbers

- If we want to find the first *n* perfect numbers, we need to count the perfect numbers we have found
    - We convert the form to a while loop
        - This means initializing and incrementing the loop variable
    - We jump out of the loop the moment we reach the correct count
        - This program will not work because it takes too much time to find the fifth perfect number

Perfect Numbers

- Why does it take so long?
  - For every number i investigated, we check $i - 2$ divisors
  - If we investigate numbers up to N, we make $\sum_{i=2}^{N} (i-2) = \frac{1}{2}\left(N^2 - 3N + 2\right)$ comparisons
    - The square kills us: if we want to go up to 10^9, we need about 10^{18} comparisons
    - If any comparison takes 10 nanoseconds, we use up 0.3 years to find the fifth perfect number

Perfect Numbers

- For the first 8, we need 1.68483×10^{21} years
  - An AWS a1 instance has 16 cores, so we need to have 1.05302×10^{20} instances for a year
  - Which costs us about 3.6923×10^{23} dollars.

Amicable Numbers

- Two numbers m and n are amicable, if the sum of divisors of m is n and the sum of divisors of n is m
  - Smallest example is 220 and 264
  - 220=1+2+3+4+6+8+11+12+22+24+33+44+66+88+132
  - 264=1+2+4+5+10+11+20+22+44+55+110

Amicable Numbers

- To find amicable numbers up to 10000:
    - Let `number` vary between 1 and 10000
    - Calculate the sum of divisors of `number`, called `sum_of_div`
    - Then calculate the sum of divisors of `sum_of_div`
    - If that number equals `number`, we have two amicable numbers

Amicable Numbers

- Since amicable numbers come in pairs, impose additional restriction $m < n$.
  - Since we start with m and calculate n, we can jump to the next value for m if $m \geq n$.

Amicable Numbers

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

Initializing number and count

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

Enter an “infinite” loop

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

Get the next number

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

Find the sum of divisors of number

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

We already now that this is not a correct pair of amicable numbers, so we can save ourselves the work

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

We already now that this is not a correct pair of amicable numbers, so we can save ourselves the work

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

Find the sum of divisors of sum_of_div

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

Hurrah: we found one

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

Hurrah: we found one
Print it out, count it

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

And if the count reaches 15, we can break

```
count = 0
number = 2
while True:
    number += 1
    sum_of_div = 0
    for j in range(1,number):
        if number%j == 0:
            sum_of_div += j
    if sum_of_div <= number:
        continue
    suma2 = 0
    for j in range(1,sum_of_div):
        if sum_of_div%j == 0:
            suma2 += j
    if suma2 == number:
        print(number, sum_of_div)
        count += 1
        if count == 15:
            break
```

Improvement: Replace 15 with a constant

Amicable Numbers

- Takes a noticeable amount of time
  - Mathematicians still investigate amicable numbers because so many things are unknown
  - But our simple “complete enumeration” algorithms will not be able to compete

```
220 284
1184 1210
2620 2924
5020 5564
6232 6368
10744 10856
12285 14595
17296 18416
63020 76084
66928 66992
67095 71145
69615 87633
79750 88730
100485 124155
122265 139815
```

Amicable Numbers

- Our code is also still clumsy
    - It is too long with too many steps of logic
    - That’s because we do not yet have the methods to break it up
- Also: if that is the best example for a real use of “continue”, it shows that “continue” is quite a bit rarer than “break”