

While Loops

Thomas Schwarz, SJ

While Loops

- While for loops are useful
 - While loops are more general:
 - Every for loop can be transformed into a while loop

While Loops

- Form of the while loop:

```
while condition :  
    Statement Block
```

← Indent →

- Keyword is while
- Condition needs to evaluate to either True or False
 - Condition is a boolean

While Loops

- Statement block is executed as long as condition is valid.
- Allows the possibility of infinite loops

```
while condition :
```

←→ Statement Block
Indent

Apple Inc.
One Infinite Loop
Cupertino, CA 95014
(408) 606-5775

An Infinite Loop

```
while True:  
    print("Hello World")
```

If this happens to you, you might

While Loops can emulate for loops

- Find an equivalent while loop for the following for-loop

- (which calculates $\sum_{\nu=1}^n \frac{1}{\nu}$)

```
n = int(input("Enter n: "))
suma = 0
for i in range(1,n+1):
    suma += 1/i
print("The", n, "th harmonic number is", suma)
```

While loops can emulate for loops

- Solution: the loop-variable i has to start out as 1 and then needs to be incremented for every loop iteration
- We stop the loop when i reaches $n+1$, i.e. we continue as long as $i \leq n$.

```
n = int(input("Enter n: "))
sum = 0
i = 1
while i <= n:
    sum += 1/i
    i += 1
print("The", n, "th harmonic number is", sum)
```

While Loops

- An old Chinese Mathematics puzzle:
 - A band of 17 pirates steal some gold coins.
 - When they divide the spoils equally, 3 coins are left over
 - Leading to a fight in which one pirate is killed
 - After settling, they try again, but now 10 coins are left
 - Another fight breaks out, killing another pirate.
 - And now the equal division works.
- What is the smallest number of gold coins?

While Loops

- Idea:
 - We try out all numbers of gold bars $x = 1, 2, 3, \dots$
 - Unlike a for-loop, we do not have to predetermine an upper value for x
 - Which leaves open the possibility of never finding something
 - Conditions are
 - $x \equiv 3 \pmod{17}$
 - $x \equiv 10 \pmod{16}$
 - $x \equiv 0 \pmod{15}$

While Loops

- Details:
 - Unlike a for-loop, **the programmer** has to initialize x and increment x in the loop

While Loops

- We also need to be able to stop the loop
 - This is done with the **break** statement
 - With the break statement, we simply “jump out” of the loop

While Loops

Initialize coins



```
coins = 1
while True:
    if coins%17 == 3 and coins%16 == 10 and coins%15 == 0:
        print("The answer is", coins)
        break
    coins += 1
```

While Loops

Initialize coins



```
coins = 1
while True:
    if coins%17 == 3 and coins%16 == 10 and coins%15 == 0:
        print("The answer is", coins)
        break
    coins += 1
```

While Loops


```
coins = 1
while True:
    if coins%17 == 3 and coins%16 == 10 and coins%15 == 0:
        print("The answer is", coins)
        break
    coins += 1
```



Checking

While Loops

```
coins = 1
while True:
    if coins%17 == 3 and coins%16 == 10 and coins%15 == 0:
        print("The answer is", coins)
        break
    coins += 1
```



Found it:
print out
break out of loop

While Loops

```
coins = 1
while True:
    if coins%17 == 3 and coins%16 == 10 and coins%15 == 0:
        print("The answer is", coins)
        break
    coins += 1
```

Next case

While Loops

- With a little bit more logic, we can avoid using the while-true-break paradigm
 - Though it is very pythonesque

While Loops

- Alternative:

```
coins = 1
while not (coins%17 == 3 and coins%16 == 10 and coins%15 == 0):
    coins += 1
print('There were', coins, 'coins.')
```

While Loops

- Alternative:



Initialization

```
coins = 1
while not (coins%17 == 3 and coins%16 == 10 and coins%15 == 0):
    coins += 1
print('There were', coins, 'coins.')
```

While Loops

- Alternative:

Test whether we are NOT done

```
coins = 1
while not (coins%17 == 3 and coins%16 == 10 and coins%15 == 0) :
    coins += 1
print('There were', coins, 'coins.')
```

While Loops

- Alternative:

```
coins = 1
while not (coins%17 == 3 and coins%16 == 10 and coins%15 == 0):
    coins += 1
print('There were', coins, 'coins.')
```



We are not done, so we go to the next one

While Loops

- Alternative:

```
coins = 1
while not (coins%17 == 3 and coins%16 == 10 and coins%15 == 0):
    coins += 1
print('There were', coins, 'coins.')
```

This is tricky:

The current value of coins is indeed the value that fulfills the condition.

While Loops

- We can try to make this easier by using a Boolean variable done

```
coins = 0
done = False
while not done:
    done = coins%17 == 3 and coins%16 == 10 and coins%15 == 0
    if done:
        print('The number of coins is', coins)
    coins += 1
```

While Loops

- We can try to make this easier by using a Boolean variable done

```
coins = 0
done = False
while not done:
    done = coins%17 == 3 and coins%16 == 10 and coins%15 == 0
    if done:
        print('The number of coins is', coins)
    coins += 1
```



Initializing

While Loops

- We can try to make this easier by using a Boolean variable done

```
coins = 0
done = False
while not done:
    done = coins%17 == 3 and coins%16 == 10 and coins%15 == 0
    if done:
        print('The number of coins is', coins)
    coins += 1
```

Recalculate the condition

While Loops

- We can try to make this easier by using a Boolean variable done

```
coins = 0
done = False
while not done:
    done = coins%17 == 3 and coins%16 == 10 and coins%15 == 0
    if done:
        print('The number of coins is', coins)
    coins += 1
```

Don't forget to increment coins
or you end up in an infinite
loop.

While Loops

- We can optimize by removing the if-statement
 - In this version, we need to take an additional increment of coins into account when done evaluates to True

```
coins = 0
done = False
while not done:
    done = coins%17 == 3 and coins%16 == 10 and coins%15 == 0
    coins += 1
print('The number of coins is', coins-1)
```

While Loops

- We can optimize by removing the if-statement
 - In this version, we need to take an additional increment of coins into account when done evaluates to True

```
coins = 0
done = False
while not done:
    done = coins%17 == 3 and coins%16 == 10 and coins%15 == 0
    coins += 1
print('The number of coins is', coins-1)
```



Initialization

While Loops

- We can optimize by removing the if-statement
 - In this version, we need to take an additional increment of coins into account when done evaluates to True

```
coins = 0
done = False
while not done:
    done = coins%17 == 3 and coins%16 == 10 and coins%15 == 0
    coins += 1
print('The number of coins is', coins-1)
```



Re-calculate the condition

While Loops

- We can optimize by removing the if-statement
 - In this version, we need to take an additional increment of coins into account when done evaluates to True

```
coins = 0
done = False
while not done:
    done = coins%17 == 3 and coins%16 == 10 and coins%15 == 0
    coins += 1
print('The number of coins is', coins-1)
```



We always increment

While Loops

- We can optimize by removing the if-statement
 - In this version, we need to take an additional increment of coins into account when done evaluates to True

```
coins = 0
done = False
while not done:
    done = coins%17 == 3 and coins%16 == 10 and coins%15 == 0
    coins += 1
print('The number of coins is', coins-1)
```

But the last time, we should
not

Harmonic Numbers

- The n th harmonic number is $h_n = \sum_{\nu=1}^n \frac{1}{\nu}$
 - It is known that this series diverges.
- Given a positive number x , we want to determine n such that the n th harmonic number is just above x

$$\min(\{n \mid h_n > x\})$$

- Solution: add $\frac{1}{\nu}$ while you have not reached x

Harmonic Numbers

```
x = float(input("Enter x: "))
nu = 1
sum = 0
while sum <= x:
    sum += 1/nu
    nu += 1
print("The number you are looking for is ", nu-1,
      "and incidentally, h_n =", sum)
```

- When we stop, we need to undo the last increment of nu, but not for sum.