



## while loop, round 1

- Top-tested loop (pretest)
  - test the boolean before running
  - test the boolean before each iteration of the loop

while boolean expression:  
statementSuite



## repeat while the boolean is true

- while loop will repeat the statements in the suite while the boolean is True (or its Python equivalent)
- If the boolean expression never changes during the course of the loop, the loop will continue forever.



## General approach to a while

- outside the loop, initialize the boolean
- somewhere inside the loop you perform some operation which changes the state of the program, eventually leading to a False boolean and exiting the loop
- Have to have both!



## Example 7

## Example 7

- Example illustrates use of repetition
- Calculates factorial:  $n!$ 
  - $1! = 1$
  - $n! = 1 * 2 * 3 * \dots * n$
- Actually calculates it in reverse order:
  - $n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$



## While example

```
intStr = raw_input('Factorial; enter an integer: ')
myInt = int(intStr)
```

```
factValue = 1
while myInt > 0:
    factValue = factValue * myInt
    myInt = myInt - 1
print factValue
```



## Counting while template

```

init the counting variable
while countingBoolean:
    suite
    suite
    lastly, update the counting variable
  
```



## while loop, round two

- while loop, oddly, can have an associated else statement
- else statement is executed when the loop finishes under normal conditions
  - basically the last thing the loop does as it exits



## while with else

```

while booleanExpression:
    suite
    suite
else:
    suite
    suite
rest of the program
  
```



## While 'early exit': break, continue

```

while test1:
    statement_list_1
    if test2: break # Exit loop now; skip else
    if test3: continue # Go to top of loop now
    # more statements
else:
    statement_list_2 # If we didn't hit a 'break'

# 'break' or 'continue' lines can appear anywhere
  
```



## Example 8

## Example: 8 test for primality

- Is positive integer N prime?
- Algorithm:
  - Test all possibilities:
    - see if any numbers divide evenly
    - if so, N is not prime
- Improvements
  - only test up to  $\sqrt{N}$
  - start at  $\sqrt{N}$  and work down one at a time
  - remember to make  $\sqrt{N}$  an integer:  $\text{int}(\sqrt{N})$



## Example 8: test for primality

```
import math # get square root function from math module

n = raw_input('Is this positive integer prime? ')
n = int(n)

d = int(math.sqrt(n)) # upper limit of possible factors of n

while d > 1:
    if n % d == 0: # evenly divisible: found factor
        print n, ' has factor ', d
        break # skip else
        d = d - 1
    else:
        print n, ' is prime. '
```



## Example 9

## Example 9: more control

Ask the user to enter a string longer than 3  
If not, make them try again.  
If they enter "quit", quit.



```
while True:
    s = raw_input('Enter a string longer than 3 ("quit" to end): ')
    if s == 'quit':
        break
    if len(s) < 4:
        continue # try again
    print 'Input is of sufficient length'
    print 'You entered: ', s
```



## Example 10 Primality with input check

```
import math # get square root function from math module

while True:
    n = raw_input('Is this positive integer prime? ')
    n = int(n)
    if n > 1:
        break
    print "Error: input not positive; try again."

d = int(math.sqrt(n)) # upper limit of possible factors of n

while d > 1:
    if n % d == 0: # evenly divisible: found factor
        print n, ' has factor ', d, 'and therefore isn't prime'
        break # skip else
        d = d - 1
    else:
        print n, ' is prime. '
```



## Example 11 Password Check

- Password Check with Three Tries
- Check to see if password is in a list
- If not, try again
- After three attempts, reject




```

Password Check with Three Attempts
valid = False
count = 3 # count of number of attempts
while count > 0:
    input = raw_input("Enter password: ")
    for passwd in passwdList: # check input against all passwords
        if input == passwd:
            valid = True
            break # break out of "for"
    if not valid:
        count = count - 1
        print "Bad password. Try again. You have", count, "tries remaining"
        continue # go directly to top of loop
    else:
        break # break out of "while"
if count > 0:
    print "Password Accepted"
else:
    print "Password Rejected"

```

Michigan State University CSE 231, Fall 2009 Repetition 25

## ...a little history



**Alan Turing,**  
[OBE \(1912 – 1954\)](#)  
 was an [English mathematician](#), [logician](#),  
 and [cryptographer](#).

Turing is often considered to be the father of modern [computer science](#).

wikipedia.org

Michigan State University CSE 231, Fall 2009 Repetition 26

Turing provided an influential formalization of the concept of the [algorithm](#) and computation with the [Turing machine](#): that any practical computing model has either the equivalent or a subset of the capabilities of a Turing machine.

With the [Turing test](#), he made a significant and characteristically provocative contribution to the debate regarding [artificial intelligence](#): whether it will ever be possible to say that a machine is [conscious](#) and can [think](#).

wikipedia.org

Michigan State University CSE 231, Fall 2009 Repetition 27

**Turing machines** are extremely basic abstract symbol-manipulating devices which, despite their simplicity, can be adapted to simulate the logic of any [computer](#) that could possibly be constructed.

They were described in [1936](#) by [Alan Turing](#).

Though they were intended to be technically feasible, Turing machines were not meant to be a practical computing technology, but a [thought experiment](#) about the limits of mechanical computation; thus they were not actually constructed.

Studying their [abstract properties](#) yields many insights into [computer science](#) and [complexity theory](#).

wikipedia.org

Michigan State University CSE 231, Fall 2009 Repetition 28

During the [Second World War](#) Turing worked at [Bletchley Park](#), Britain's [codebreaking](#) centre, and was for a time head of [Hut 8](#), the section responsible for [German](#) naval cryptanalysis.

He devised a number of techniques for breaking German ciphers, including the method of the [bombe](#), an electromechanical machine that could find settings for the [Enigma machine](#).

wikipedia.org

Michigan State University CSE 231, Fall 2009 Repetition 29

## Mark I



In 1947 Turing moved to the [University of Manchester](#) to work, largely on software, on the [Manchester Mark I](#), then emerging as one of the world's earliest computers.

wikipedia.org

Michigan State University CSE 231, Fall 2009 Repetition 30

## Mark I specs

- Store organised in 20-bit addressable lines, an instruction taking one line and a number two consecutive lines
- Serial 40-bit arithmetic, with hardware add, subtract and multiply (with a double-length accumulator) and logical instructions
- 8 modifier registers (B-lines for modifying addresses in instructions); simple B-line arithmetic and tests
- Single-address format order code with about 50 function codes.
- 8 pages of random access main store (one CRT per 64 \* 20-bit line page)
- 512 page capacity drum backing store, 2 pages per track, about 30 ms revolution time
- Standard instruction time: **1.2 ms, multiplication in 2.16 ms**
- Peripheral Instructions: read and punch a line of 5-hole paper tape;



## Review

```
while test1:
    statement_list_1
    if test2: break # Exit loop now; skip else
    if test3: continue # Go to top of loop now
    # more statements
else:
    statement_list_2 # If we didn't hit a 'break'

# 'break' or 'continue' lines can appear anywhere
```

