Quick Recap of Selection

CSE 231, Rich Enbody

Python Selection

```python
if condition1:
    suite1
elif condition2:
    suite2
elif condition2:
    suite3
...
else:
    suite_lst
```
Repetition

CSE 231, Rich Enbody

Sequential Execution

Statement 1

Statement 2

...

Statement n
Selective Execution (review)

- True: Boolean expression
- False: Boolean expression

- Statement 1
- Statement 2

Repetition

Behavior: repeat a statement

*while* some condition is True
while <condition>:
  <suite>

Remember Compound Statement

expression (optional)

keyword expression:
  statement
  statement
  statement

ends all headers

"suite" of statements

indentation under the keyword
**while** loop, round 1

Top-tested loop (pretest)
- test the Boolean *before* each iteration of the loop

```
while Boolean_expression:
    statement_suite
```

**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 (*infinite loop*)
General approach to a while

- Before the loop, initialize the Boolean.
- Somewhere inside the loop, change the Boolean so the loop will exit.
- Need both!

While example

```
n = input(...)  
n = int(n)  
f = 1  
while n > 0:  
    f = f * n  
    n = n - 1  
print (f)
```
While example

```
n = input(...)  # input n
n = int(n)     # convert n to integer
f = 1

n > 0:
    f = f * n  # calculate f
    n = n - 1
    print(f)  # print f
```

Example 7
Example 7

- Example illustrates use of repetition
- Calculates factorial: n!
  \[1! = 1\]
  \[n! = 1 \times 2 \times 3 \times \ldots \times n\]
- Actually calculates it in reverse order:
  \[n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1\]

```python
while example

num_str = input('Factorial; enter an integer: ')
num_int = int(num_str)

factorial_int = 1
while num_int > 1:
    factorial_int = factorial_int * num_int
    num_int = num_int - 1
print(factorial_int)
```
Counting while template

init the counting variable
while counting_Boolean:
    suite
    last, update the counting variable
Exercise 1

# exercise 1
# print integers from 3 to 7 inclusive

# figure out INITIAL_VALUE, BOOLEAN_EXPRESSION, and CHANGE_i

i = INITIAL_VALUE

while BOOLEAN_EXPRESSION:
    print(i)
    CHANGE_i

# Bonus: why can't you do "print real numbers from 3.0 to 7.0 inclusive"?

while loop, round two

- while loop with else statement
- else statement is executed when the loop finishes under normal conditions:
  basically the last thing the loop does as it exits
while Boolean_expression:
    suite
else:
    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: int(sqrt(N))

Example 8: test for primality

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

num_str = input('Is this positive integer prime? ')  
num_int = int(num_str)

divisor_int = num_int - 1  
# divisor_int = int(math.sqrt(num_int ))  # upper limit of possible factors of n

while divisor_int > 1:
    if num_int % divisor_int == 0:  # evenly divisible: found factor
        print(num_int , ' has factor ', divisor_int)
        break  # skip else
    divisor_int = divisor_int - 1
else:
    print (num_int , ' is prime. ')
```
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.
How does the loop end?

```python
while True:
    s = input('Enter a string longer than 3 ("quit" to end): ')
    if s == 'quit':
        break
    if len(s) < 4:
        print('No, ', s, ' is not long enough. ')
    continue  # try again
    print('Yes. Input', s, 'is of sufficient length')

print('Thanks for playing')
```

Example 10
Example 10 Primality with input check

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

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

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

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

Exercise 2
Exercise 2

# exercise 5-2
# Take an integer n as input; print the integers from 0 to n that are divisible by 3
# Find: INITIAL_VALUE, boolean_expression_1, boolean_expression_2, change_i
n_str = input("Enter an integer: ")
n_int = int(n_str)

i = initial_value
while boolean_expression_1:
    if boolean_expression_2:
        print(i, end = ' ')
    change_i

# Bonus: print a count of how many numbers you printed

Example 11
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

```python
valid = False
count = 3  # count of number of attempts
while count > 0:
    input_str = input("Enter password: ")
    for passwd in passwdList:  # check input against all passwords
        if input_str == 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 while loop
    else:
        break  # break out of "while"
if count > 0:
    print("Password Accepted")
else:
    print("Password Rejected")
```
…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.

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.
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.

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.
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.

---

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