CHAPTER 5
LOOPS

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING USING PYTHON, LIANG (PEARSON 2013)
CONTROL FLOW

• Control flow.
  • Sequence of statements that are actually executed in a program.
  • Conditionals and loops: enable us to choreograph control flow.
MOTIVATIONS

• Suppose that you need to print a string (e.g., "Welcome to Python!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```python
print("Welcome to Python!")
```

So, how do you solve this problem?

• How about altering our guessing game program to allow 20 tries?
OPENING PROBLEM

print("Welcome to Python!")
print("Welcome to Python!")
print("Welcome to Python!")
print("Welcome to Python!")
print("Welcome to Python!")
print("Welcome to Python!")
print("Welcome to Python!")

...
print("Welcome to Python!")
print("Welcome to Python!")
THE WHILE LOOP
INTRODUCING WHILE LOOPS

1. count = 0
2. while count < 100:
3.   print("Welcome to Python")
4.   count += 1
WHILE LOOP FLOW CHART

1. while loop-continuation-condition:
2. # loop-body
3. Statement(s)

1. count = 0
2. while count < 100:
3. print("Welcome to Python")
4. count += 1
TRACING WHILE LOOPS

1. count = 0
2. while count < 2:
3.   print("Welcome to Python")
4.   count += 1
1. `count = 0`  
2. `while count < 2:`  
3. `print("Welcome to Python")`  
4. `count += 1`
TRACING WHILE LOOPS

1. count = 0
2. \texttt{while count < 2:
3. \phantom{b} \texttt{print("Welcome to Python")
4. \phantom{b} \texttt{count += 1

\textbf{Memory}
\begin{align*}
\text{count: 0}
\end{align*}

\textbf{Output}
\hfill
TRACING WHILE LOOPS

1. count = 0
2. while count < 2:
3.   print("Welcome to Python")
4.   count += 1

Output
Welcome to Python

Memory
count: 0

Output
Welcome to Python
TRACING WHILE LOOPS

1. count = 0
2. while count < 2:
3.   print("Welcome to Python")
4.   count += 1

Memory
count: 0 1

Output
Welcome to Python
TRACING WHILE LOOPS

1. count = 0
2. while count < 2:
   3. print("Welcome to Python")
4. count += 1

Memory
count: 0 1

Output
Welcome to Python
1. count = 0
2. while count < 2:
3.   print("Welcome to Python")
4.   count += 1

Memory
count: 0 1

Output
Welcome to Python
Welcome to Python
TRACING WHILE LOOPS

1. count = 0
2. while count < 2:
3.   print("Welcome to Python")
4.   count += 1

Memory
count: 0 \+ 2

Output
Welcome to Python
Welcome to Python

Increment count
TRACING WHILE LOOPS

1. count = 0
2. while count < 2:
3.   print("Welcome to Python")
4.   count += 1

Memory
count: 0 + 2

Output
Welcome to Python
Welcome to Python

Count < 2 is false
TRACING WHILE LOOPS

1. `count = 0`
2. `while count < 2:`
3. `print("Welcome to Python")`
4. `count += 1`

Memory
- count: 0 + 2

Output
- Welcome to Python
- Welcome to Python

Continue after
EXAMPLES – WITH A PARTNER

• What are the values of $n$ and $m$ after this program:
  
  ```python
  n = 1234567
  m = 0
  while n != 0:
      m = (10*m) + (n % 10)
      n //= 10
  ```

• Show the trace of the program at each step
QUESTION

• What is wrong with the following code?
• What happens?
• Fix it and explain what the code outputs

1. \( i, N = 0, 10000 \)
2. \( \text{while} \ i \leq N: \)
3. \( \text{print}(i) \)
4. \( i = i + 5 \)
ACTIVITY

• Write an algorithm to compute the number of digits an integer has.
  • Example: input – 34567 output – 5

• Bonus: modify your algorithm to compute the number of “digits” that the
  number would have if converted to another base, e.g., binary, octal, or
  hexadecimal
CAUTION

• Don’t use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results. Consider the following code for computing $1 + 0.9 + 0.8 + \ldots + 0.1$:

1. item, sum = 1, 0
2. while item != 0:  # No guarantee item will be 0
3.    sum += item
4.    item -= 0.1
5. print(sum)
THE FOR LOOP

```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.");
    return 0;
}
```
FOR LOOPS

1. for var in sequence:
2.   # loop body
3.   Statement(s)

Example
1. for x in range(0, 100):
2.   print("Welcome to Python!")

Initialize var to first element

Have all elements in sequence been visited?

Try to update var to be the next element in the sequence

True

Statement(s)

False

Here "end" refers to 1 after the last element of the sequence.
TRACING FOR LOOPS

1. for x in range(0, 2):
2.   print("Welcome to Python!")
TRACING FOR LOOPS

1. for \texttt{x} in \texttt{range}(0, 2):
2. \texttt{print}("Welcome to Python!")

\begin{align*}
\text{Memory} & \\
\text{x: 0} & \\
\text{*Note* range(0, 2) is [0, 1]} &
\end{align*}

\begin{align*}
\text{Output} & 
\end{align*}
TRACING FOR LOOPS

1. for \( x \) in range(0, 2):
2. print("Welcome to Python!")

*Note* range(0, 2) is [0, 1]

Memory
x: 0

Have all elements been visited? No

Output
TRACING FOR LOOPS

1. for x in range(0, 2):
2.    print("Welcome to Python!")

Memory
x: 0

*Note* range(0, 2) is [0, 1]

Output
Welcome to Python!
TRACING FOR LOOPS

1. for \(x\) in range(0, 2):
2. print("Welcome to Python!"")

*Note* range(0, 2) is \([0, 1]\)
TRACING FOR LOOPS

1. for \texttt{x in range}(0, 2):

2. \texttt{print}("Welcome to Python!")

Memory
\begin{align*}
x : & \ 0 \ 1 \\
\text{*Note* range}(0, 2) \ & \text{is } [0, 1]
\end{align*}

Output
Welcome to Python!

Have all elements been visited? No
TRACING FOR LOOPS

1. for x in range(0, 2):
2.   print("Welcome to Python!")

Memory
x: 0 1

*Note* range(0, 2) is [0, 1]

Output
Welcome to Python!
Welcome to Python!
TRACING FOR LOOPS

1. for $x$ in range(0, 2):
2.   print("Welcome to Python!")

*Note* range(0, 2) is [0, 1]

Memory
x: 0 1

Output
Welcome to Python!
Welcome to Python!
1. for \( x \) in `range(0, 2)`:
   
2. `print("Welcome to Python!")`

*Note* `range(0, 2)` is \([0, 1]\)
TRACING FOR LOOPS

1. for x in range(0, 2):
2.   print("Welcome to Python!")

Memory
x: 0 1

*Note* range(0, 2) is [0, 1]

Output
Welcome to Python!
Welcome to Python!

Continue after
Range is a method that generates a sequence of integer numbers:

- \texttt{range}(a, b, \text{step}) – generates numbers from \textit{a} up to but not including \textit{b} with an increment of \textit{step}, e.g., \texttt{range}(2, 10, 2) returns [2, 4, 6, 8]
- \texttt{range}(a, b) – generates numbers from \textit{a} up to but not including \textit{b} with an increment of 1, e.g., \texttt{range}(1, 5) returns [1, 2, 3, 4]
- \texttt{range}(b) – generates numbers between 0 and \textit{b} with an increment of 1, e.g., \texttt{range}(3) returns [0, 1, 2]
PRACTICE

• Group 1: Write a for loop to output all numbers between integers $a$ and $b$
• Group 2: Write a for loop to output the multiples of an integer $a$ up to $N$
• Group 3: Write a for loop to output all the even numbers from 100 to 999 in reverse order.
COMPARE FOR LOOPS TO WHILE LOOP

1. count = 0
2. while count < 100:
   3. print("Welcome to Python")
   4. count += 1

1. for x in range(1, 100):
   2. print("Welcome to Python!")

Note, each has their own use. For loops are a special case in which each element of a sequence is visited. In this case (and only this case) are for-loops appropriate in Python.
NESTING

• In control flow, nesting is where you place a control structure inside of another
• Example: 2 for loops to print a multiplication table

1. for i in range(0, 10):
2.   for j in range(0, 10):
3.     print(str(i) + "*" + str(j) + " = "
       + format(i*j, "2d"), end="  ")
4.   print() # Print a new line
EXERCISE – FIX THE GUESSING GAME

• Lets fix our guessing game program to allow up to 20 guesses. Additionally, try to protect against bad input

• Program this together

• If you get lost program is on following slides (split into multiple slides)
EXERCISE – WHERE TO BEGIN

• When developing programs
  • Always think first!
  • Sketch out solution, i.e., plan
  • Implement solution
  • Test solution
  • Repeat!

• Called iterative development
EXERCISE – FIX THE GUESSING GAME

1. import random
2.
3. # Grab a random number
4. rn = random.randint(1, 99)
5. un = 0
6. guesses = 0
7.
8. # Allow user to continually guess
9. while rn != un and guesses < 20:
10.     un = int(input("Please enter a number between 1 and 99: "))
11.
12. if un < 1 or un > 99:
13.     print("Invalid guess.")
14. elif un == rn:
15.     print("You won!")
16. elif un > rn:
17.     print("Too high")
18.     guesses += 1
19. else: # un < rn
20.     print("Too low")
21.     guesses += 1
22.
23. if guesses == 20:
24.     print("You lost. Out of guesses. The correct number is " + str(rn) + ".")
MONTE CARLO SIMULATION
GAMBLER'S RUIN

• Gambler's ruin. Gambler starts with $stake and places $1 fair bets until going broke or reaching $goal.
  • What are the chances of winning?
  • How many bets will it take?

• One approach. Monte Carlo simulation.
  • Flip digital coins and see what happens.
  • Repeat and compute statistics.
GAMBLER'S RUIN

1. import random
2.
3. stake, goal, T = eval(input("Enter stakes, goal, and T: "))
4.
5. wins = 0
6. for t in range(T):
    cash = stake
8.     while cash > 0 and cash < goal:
9.         if random.random() < 0.5:
10.             cash += 1
11.         else:
12.             cash -= 1
13.     if cash == goal:
14.         wins += 1
15.
16. print(wins, "wins of", T)
OTHER CONTROL FLOW STATEMENTS
OTHER HELPFUL STATEMENTS FOR LOOPS

- **break** – immediately exit the loop. Do not continue executing any more of the loop. Example:

  ```python
  while True:
      if q-key-is-pressed():
          # quit the game
          break
  Game-loop()
  ```

- **continue** – immediately skip to the end of the body of the loop, i.e., start next iteration. Example:

  ```python
  for i in range(0, 10):
      if isPrime(i)
          # OCD against prime numbers
      continue
  HandleNotPrimes()
  ```
CONTROL FLOW SUMMARY

- Control flow.
  - Sequence of statements that are actually executed in a program.
  - Conditionals and loops: enable us to choreograph the control flow.

<table>
<thead>
<tr>
<th>Control Flow</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight-line programs</td>
<td>All statements are executed in the order given</td>
<td></td>
</tr>
<tr>
<td>Conditionals</td>
<td>Certain statements are executed depending on the values of certain variables</td>
<td>if; if-else</td>
</tr>
<tr>
<td>Loops</td>
<td>Certain statements are executed repeatedly until certain conditions are met</td>
<td>while; for</td>
</tr>
</tbody>
</table>
EXERCISE

• Write a program to draw a checkerboard pattern with Turtle (either a Checker's board or a Chess board)
  • You can set the speed of the turtle to infinity (turtle.speed(0))
  • Turtle allows the ability to draw a filled rectangle with turtle.begin_fill() and turtle.end_fill()