COMP1021
Introduction to Computer Science

Loops

David Rossiter and Gibson Lam

Outcomes

- After completing this presentation, you are expected to be able to:
  1. Write loops using the while command
  2. Work with conditions using logical operators
  3. Write code using nested loops

Loops

- Using loops in programming is very useful because it makes repetitive work easy
- In this presentation we look at *while* loops
- We will use both graphics and non-graphics examples

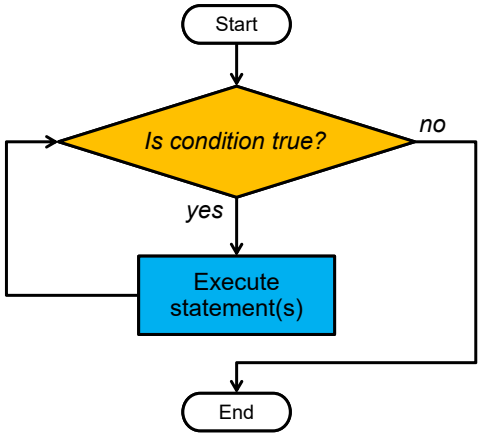
While Loops

`while` ...*condition*... **:**

...*statement(s)*...

- While *condition* is true, repeatedly execute *statement(s)*
  - A statement simply means a Python instruction
- When *condition* is false, the while loop finishes

The Flow of a While Loop



Reminder - Comparison

- You can do the following comparisons:

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Counting Up

- This example counts from 1 to 10
- Each time it prints the number

```
count = 1
while count <= 10:
    print(count)
    count = count + 1
```

When the program is executed, this is what you see

```
1
2
3
4
5
6
7
8
9
10
```

Like the Python `if` *statement, we need to use indentation for everything inside the* `while`

1. The value `1` *is put in the control variable* `count`

2. If the value in `count` *is* `<= 10` *then do the things inside the loop*

3. Inside the loop the number inside the variable is printed, then it is increased by 1

4. When Python has finished doing the things inside the while loop, it will automatically jump back to the `while` *and check whether to do the things inside the loop again*

Result:

```
1
2
3
4
5
6
7
8
9
10
```

Counting Down

- This example does the opposite to the previous example
- This time it counts down, from 10 to 1

```
count = 10
while count >= 1:
    print(count)
    count = count - 1
```

When the program is executed, this is what you see

```
10
9
8
7
6
5
4
3
2
1
```


What Happens When a Loop Finishes?

- When a loop finishes Python simply goes to the next line of code after the loop, and carries on

```
count = 10
while count >= 1:
    print(count)
    count = count - 1
print("finished!")
```

When the program is executed, this is what you see

```
10
9
8
7
6
5
4
3
2
1
finished!
```

You know this is not in the loop because there's no indentation

Writing Comments

- Python will ignore anything on the right of #
- So you can use it to make notes, like this:

```
# This is an example of a loop
# It will count down from 10 to 1
count=10 # Start with the number 10
while count>=1:
    print(count) # Show the number
    count=count-1 # Decrease the variable
```


Another Way to Do Comments

- When you want to write a big comment, you can use """ at the start and end, instead of starting every line of your comment with a #

```
"""
This is an example of a loop.
It counts down from 10 to 1.
Each time it prints the number.
"""
```

- (However, sometimes Python gets a bit confused when you use this method, the # method is safer)

Using Loops For Graphics

- Loops are very useful for graphics because many graphical structures are created by repeating code
- For example, to draw a square you can *move forward and change angle 90 degrees* four times, as shown here:

```
import turtle
. . .
turtle.forward(200)
turtle.right(90)
turtle.forward(200)
turtle.right(90)
turtle.forward(200)
turtle.right(90)
turtle.forward(200)
turtle.right(90)
```

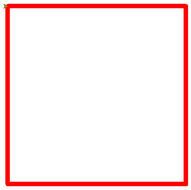
Drawing a Square

- This code uses a loop to create the same square

In this presentation we are focused on loops so we don't show the first few turtle commands e.g.

```
import turtle
turtle.color("red")
```

```
side = 0
while side < 4:
    turtle.forward(200)
    turtle.right(90)
    side = side + 1
```

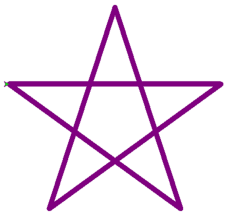


Run the loop four times
i.e. the loop will be executed with the variable `side` *containing* 0, 1, 2, and 3

Drawing a Star Shape

- Similarly you can use a loop to draw a star shape with five sides, i.e.:

```
side = 0
while side < 5:
    turtle.forward(200)
    turtle.right(144)
    side = side + 1
```



Run the loop five times
i.e. the loop will be executed with the variable `side` *containing* 0, 1, 2, 3, and 4

Another Example

- In this example the value in a variable called *radius* is reduced each time
- The variable is used to control the radius of a circle
- So the circle gets smaller each time

Repeat the loop while the radius is greater than zero

```
radius = 100
while radius > 0:
    turtle.circle(radius)
    radius = radius - 10
```

An Eating Candy Example

- The program below uses a while loop to repeatedly buy candy bars while there's enough money

Start with this much money in the pocket

The loop runs while there is enough money to buy a candy bar

```
money_in_pocket = 30
cost_of_candy_bar = 7

while money_in_pocket >= cost_of_candy_bar:
    print("I have $", money_in_pocket)
    print("I am buying and eating a delicious candy bar!")

    money_in_pocket = money_in_pocket - cost_of_candy_bar

print("Now, I only have $", money_in_pocket, "left.")
print("I don't have enough money for any more candy :(")
```

Running the Eating Candy Example

- Here is the result of running the program

In this example, $7 has been used to buy one candy bar each time, inside the while loop

```
>>>
I have $ 30
I am buying and eating a delicious candy bar!
I have $ 23
I am buying and eating a delicious candy bar!
I have $ 16
I am buying and eating a delicious candy bar!
I have $ 9
I am buying and eating a delicious candy bar!
Now, I only have $ 2 left.
I don't have enough money for any more candy :(
>>>
```

Improving the Example

- Let's improve the eating candy example to include the number of candy bars that are bought
- First, a variable to count the number of candy bars is added at the top of the program, like this:

```
candy_bars_eaten = 0
```

- Then inside the while loop, the variable is increased by one, like this:

```
candy_bars_eaten = candy_bars_eaten + 1
```


The Improved Program

```
money_in_pocket = 30
cost_of_candy_bar = 7
candy_bars_eaten = 0
while money_in_pocket >= cost_of_candy_bar:
    print("I have $", money_in_pocket)
    print("I am buying and eating a delicious candy bar!")
    money_in_pocket = money_in_pocket - cost_of_candy_bar
    candy_bars_eaten = candy_bars_eaten + 1
print("I have eaten", candy_bars_eaten, "candy bars.")
print("Now, I only have $", money_in_pocket, "left.")
print("I don't have enough money for any more candy :(")
```

These are newly added code

Running the Improved Example

```
>>>
I have $ 30
I am buying and eating a delicious candy bar!
I have $ 23
I am buying and eating a delicious candy bar!
I have $ 16
I am buying and eating a delicious candy bar!
I have $ 9
I am buying and eating a delicious candy bar!
I have eaten 4 candy bars.
Now, I only have $ 2 left.
I don't have enough money for any more candy :(
>>>
```

A new message

A Math Question Example

- Here a math question is created and shown
- The user has to answer it correctly

```
import random

number1 = random.randint(1, 99)
number2 = random.randint(1, 99)

answer = number1 + number2
guess = 0

while guess != answer:
    print("What is", number1, "+", number2)
    guess = input("? ")
    guess = int(guess)

print("You are right!")
```

Generate two random numbers between 1 and 99

The user guesses the answer inside the while loop

Running the Math Question Example

- To finish the program the user has to enter the correct answer
- This is because the while loop continues when `guess` is not equal to `answer`
- In other words, `guess` must be equal to `answer` to finish the program
- Here is an example of running the program:

```
>>>
What is 28 + 75
? 100
What is 28 + 75
? 110
What is 28 + 75
? 103
You are right!
>>>
```

Using a Loop Inside a Loop

- You can put a loop inside a loop

```
start outer loop
    start inner loop
        ...statement(s)...
    end inner loop
end outer loop
```

- For example, you can put a while loop inside another while loop
- A loop inside a loop is called a *nested* loop

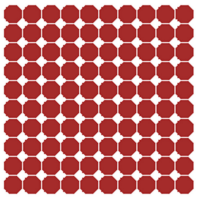
This is the target result

- Let's imagine we need to create this 10*10 pattern
- We could use two loops, one inside the other:
  - The outside loop goes from bottom to top
  - The inside loop goes from left to right, creating a circle each time
- An example implementation is shown on the next slide


```
import turtle
turtle.color("brown")
turtle.speed(0) # Fast
turtle.up() # No lines

y=0
while y<10:
    x=0
    while x<10:
        display_x=x*20
        display_y=y*20
        turtle.goto(display_x, display_y)
        turtle.dot(20)
        x=x+1

    y=y+1

print("finished!")
```

- `turtle.dot()` makes a filled circle
- The turtle position is the circle center
- It works even if the pen is up

Result:



*The result is a 10*10 display of circles*

Using an Infinite While Loop

- The previous math question program asks a question only once
- Now we change the program so that it asks math questions indefinitely
- We do this by using an *infinite loop*
- An infinite loop is a loop that never stops, e.g. the condition is always true, like this: `while True:`

...statement(s)...

A Nested Loop Example

This code asks random math questions indefinitely, because this outside loop never stops

```
import random

while True:
    number1 = random.randint(1, 99)
    number2 = random.randint(1, 99)

    answer = number1 + number2
    guess = 0

    while guess != answer:
        print("What is",number1,"+",number2)
        guess = input("? ")
        guess = int(guess)

    print("You are right!")
```

Running the Program

```
What is 10 + 63
? 74
What is 10 + 63
? 73
You are right!
What is 52 + 79
? 132
What is 52 + 79
? 130
What is 52 + 79
? 131
You are right!
What is 3 + 2
? 4
What is 3 + 2
? 5
You are right!
What is 85 + 98
? 185
What is 85 + 98
? 183
You are right!
```

```
What is 77 + 27
? 97
What is 77 + 27
? 107
What is 77 + 27
? 104
You are right!
What is 3 + 54
? 57
You are right!
What is 37 + 13
? 49
What is 37 + 13
? 50
You are right!
What is 97 + 41
?
```

Stopping the Example

- The program will not stop asking you math questions (because of the infinite loop!)
- One way to stop the program is by pressing *Control-C*, like this:

```
>>>
What is 78 + 50
? 128
You are right!
What is 55 + 42
? 97
You are right!
What is 8 + 97
? 105
You are right!
What is 19 + 77
?
Traceback (most recent call last):
  File "C:\06_while_loop_math_question_repeat_indefinite.py", line 21, in <modul
e>
    guess = input("? ") # Get the user input and store it
KeyboardInterrupt
>>>
```

Instead of answering the question, the user pressed *Control-C* here

Improving the Example

- It is not very nice when the user has to use *Control-C* to stop a program
- Let's use more sensible control in the outer loop
- Now we will only ask three different math questions in the program
- To do that, we use a variable to keep track of the number of questions the user has answered correctly so far

The Improved Example

```
import random

number_of_questions_so_far = 0

while number_of_questions_so_far < 3:
    number1 = random.randint(1, 99)
    number2 = random.randint(1, 99)

    answer = number1 + number2
    guess = 0

    while guess != answer:
        print("What is", number1, "+", number2)
        guess = input("? ")
        guess = int(guess)

    print("You are right!")

    number_of_questions_so_far = number_of_questions_so_far + 1
```

Keep track of the number of questions answered so far

Increase the number of questions answered so far

Running the Improved Example

```
>>>
What is 27 + 20
? 47
You are right!
What is 30 + 30
? 60
You are right!
What is 44 + 37
? 77
What is 44 + 37
? 71
What is 44 + 37
? 81
You are right!
>>>
```