



# CSCI 141

Lecture 10:  
Modules, random, objects, Turtles

# Announcements

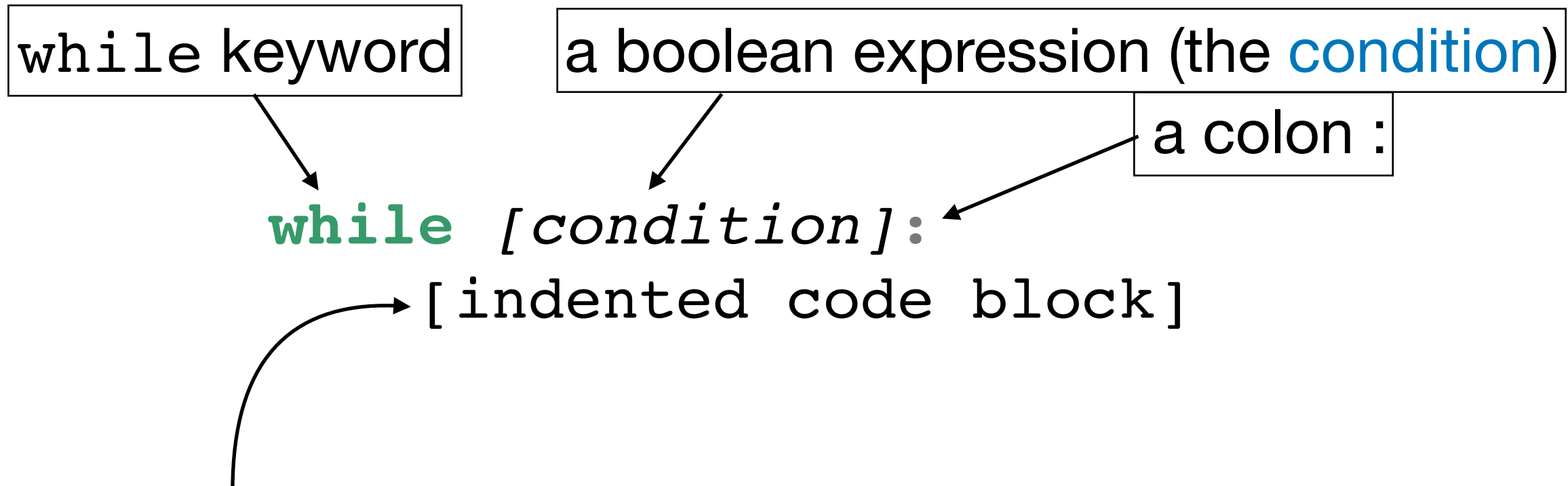
- A3 is out! Due next Wednesday.
  - Start early so you have time to study for...
- The midterm exam is a week from Friday!
  - Covers material through Monday.

# Goals

- Be able to write while loops to perform repetitive tasks, including nested while loops.
- Know how to use `import` statements to get access to `modules` containing functions that other people have written.
  - Know how to use the `random` module's `randrange` function.
- Understand how to create a Turtle `object` and call its methods on it to move it around the screen and draw shapes.

# Last time: the `while` statement

Not so different from an `if` statement:



an indented `code block`: one or more statements to be executed **while** the boolean expression evaluates to `True`



# The `while` statement:

## A Working Example

```
# print account balance after each
# of five years:
balance = 100.0 # starting balance
year = 1
while year <= 5:
    balance = balance + (0.02 * balance)
    print(balance)
    year = year + 1
```

Terminology notes:

- the line with `while` and the condition is the **loop header**
- the code block is the **loop body**
- the entire construct (header and body) is a **while statement**
- usually people call them **while loops** instead

# Warmup

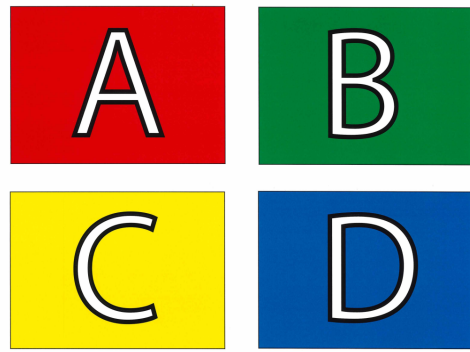


**Exercise 1:** Write a while loop to repeatedly prompt the user for a password until they get it correct.

## **Pseudocode:**

1. Ask user for password
2. If correct, go to step 3, otherwise start back at step 1
3. Print a message saying “You’re in!”

# Nesting while loops



What does this program print?

```
i = 1
while i < 4:
    j = 1
    while i * j < 6:
        print(i * j, end=" ")
        j += 1
    print()
    i += 1
```

just another statement!

A:

1	2	3	4	5	6
2	4	6			
3	6				

B:

1	2	3	4	5
2	4			
3				

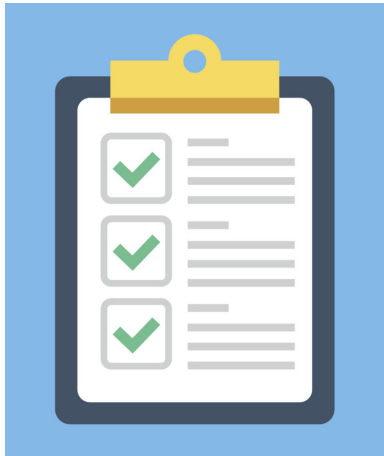
C:

1	2	4	6	2	4	3
---	---	---	---	---	---	---

D:

2	4	6
2	4	
2		

# Nesting while loops



## Exercise 2:



Print out all possible rolls of two six-sided dice.

**Program output:**

1 1  
1 2  
1 3  
1 4  
1 5  
1 6  
2 1  
2 2  
2 3  
2 4  
...  
6 4  
6 5  
6 6

(and so on)



# Nesting while loops



## Exercise 2:



Print out all possible rolls of two six-sided dice.

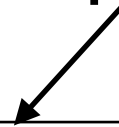
Program output:

```
1 1
1 2
1 3
1 4
1 5
1 6
2 1
2 2
2 3
2 4
...
6 4
6 5
6 6
```

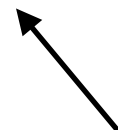
Break down the problem:

- print 1 followed by each of 1 to 6
- print 2 followed by each of 1 to 6
- and so on

Repetitive task



Repetitive task



(and so on)

# Questions?

# Other Peoples' Code

We've already used code other people wrote by calling built-in Python functions:

- `print, input, type`

Built-in functions are special because they're always available.

Many other functions exist in the Python Standard Library, which is a collection of **modules** containing many more functions.

# Other Peoples' Code

An example: I want to generate a random integer between 0 and 10.

**I don't know how to do this.**

Someone who does has written some functions for me. They live in the `random` module:

```
import random
```

I could go look at the source code...

Ar  
int  
  
I d  
  
Sc  
Tr  
  
I c

```
197  
198 ## ----- integer methods -----  
199  
200 def randrange(self, start, stop=None, step=1, _int=int):  
201     """Choose a random item from range(start, stop[, step]).  
202  
203     This fixes the problem with randint() which includes the  
204     endpoint; in Python this is usually not what you want.  
205  
206     """  
207  
208     # This code is a bit messy to make it fast for the  
209     # common case while still doing adequate error checking.  
210     istart = _int(start)  
211     if istart != start:  
212         raise ValueError("non-integer arg 1 for randrange()")  
213     if stop is None:  
214         if istart > 0:  
215             return self._randbelow(istart)  
216             raise ValueError("empty range for randrange()")  
217  
218     # stop argument supplied.  
219     istop = _int(stop)  
220     if istop != stop:  
221         raise ValueError("non-integer stop for randrange()")  
222     width = istop - istart  
223     if step == 1 and width > 0:  
224         return istart + self._randbelow(width)  
225     if step == 1:  
226         raise ValueError("empty range for randrange() (%d, %d, %d)" % (istart, istop, width))  
227  
228     # Non-unit step argument supplied.  
229     istep = _int(step)  
230     if istep != step:  
231         raise ValueError("non-integer step for randrange()")
```

e.

# Other Peoples' Code

An example: I want to generate a random integer between 0 and 10.

**I don't know how to do this.**

Someone who does has written some functions for me. They live in the `random` module:

```
import random
```

I could go look at the source code... but I'd rather just use their functions without knowing **how** they work.

```
num = random.randint(0, 10)
```



# Other Peoples' Code

```
import random  
num = random.randint(0, 10)
```

Two questions:

1. **What is this syntax about?**
2. How do I know what the function does?

# Using Modules: Syntax

The Python Standard Library is a collection of **modules** containing many more functions.

To use functions in a module, you need to **import** the module using an **import statement**:

```
import module
```

(replace the *in this font* with the specific module name)

By convention, we put all import statements at the **top** of programs.

# Using Modules: Syntax

Once you've imported a module:

```
import random
```

you can call functions in that module using the following syntax:

random.randint(0, 10)

Module name      Dot      Function call (the usual syntax)

# Other Peoples' Code

```
import random  
num = random.randint(0, 10)
```

Two questions:

1. What is this syntax about?
- 2. How do I know what the function does?**

# Other Peoples' Code

```
import random  
num = random.randint(0, 10)
```

Two questions:

1. What is this syntax about?
- 2. How do I know what the function does?**

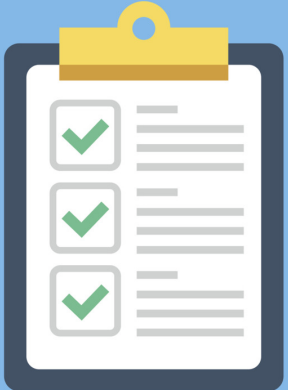
Read about it in the Python documentation.

My approach, in practice:

1. Google “python 3 <whatever>”
2. Make sure the URL is from [python.org](https://python.org) and has version python 3.x

example

# You try it



**Exercise 3:** write a program that generates and prints random integers between 1 and 10 (inclusive) until one of the random numbers exceeds 8.

Documentation says:

`random.randint(a, b)`

Return a random integer  $N$  such that  $a \leq N \leq b$



# More on import statements

- Import the entire module:

```
import random  
num = random.randint(1, 10)
```

- Import a specific function:

```
from math import sin  
sin0 = sin(0)
```

- Don't need module name dot notation
- Other random methods are not accessible

# math module

- The math module has useful stuff!
- You can read about it in the [documentation](#).
- logarithms, trigonometry, ...
- Modules can also contain values:

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>>
```

# turtle module

Python has Turtles!

```
import turtle
```



# turtle module

Python has Turtles!

```
import turtle  
scott = turtle.Turtle()
```



What does this do?  
Let's play with it.

# Demo: basic turtle usage

# Demo: basic turtle usage

- forward
- turn
- pendown/down
- penup/up



# Creating and Using Objects

```
import turtle  
scott = turtle.Turtle()
```

The `Turtle()` function starts with a capital letter.

By convention this indicates that it is a special kind of function called a **constructor** that creates (and returns) new **objects** of type `Turtle`.

The `Turtle()` function returns a `Turtle` object, and the variable `scott` now refers to it.

*functions that belong to an object are called its **methods***

Objects can have functions associated with them, accessed via the dot notation, e.g.:

```
turtle.forward(10) # moves the turtle forward 10 units  
turtle.left(90) # turns the turtle left 90 degrees
```

What methods do Turtles have? Lots!

Check the docs: <https://docs.python.org/3.3/library/turtle.html?highlight=turtle>