Middlebury

CSCI 146: Intensive Introduction to Computing

Fall 2025

Lecture 3: Style, modules and loops

Goals for today

- Distinguish between the return value of a function and its side-effects.
- Use comments and docstrings to explicate code.
- Obtain the documentation for functions and other Python objects.
- Describe aspects of good coding style.
- Explain the purpose of modules, and use standard modules like math.
- Introduce the `turtle` module.

To follow along: create a `class03.py` script where we will write code together.
(in your `cs146` folder)

First, a quick note about the modulus operator (`%`).

`print` versus `return`: I really wish `print` was called "log" or "display" instead.

`print` is really just for humans to see intermediate values as the program runs. But it has no effect on the computer running the code.

```python
def double_it(x):
    return x * 2

def another_double_it(x):
    print(x * 2)
```

Things to try:

- Add these to your `class03.py` script (with 🚀) and then let's call these in the shell.
- Now, let's try calling these directly from the script. What's the difference?

Question 1: Which of the following assigns 5 to `x`?

```python
def f1():
        return 5
def f2():
        print(5)
def f3():
        return print(5)
```

A. `x = f1()`
B. `x = f2()`
C. `x = f3()`
D. Two of A-C
E. None of the above.

Question 2: If you execute the code above as a program (with the), what will be displayed in the shell?

```python
def triangle_area(base, height):
    return base * height / 2.0
    print("Base is:", base)
triangle_area(4, 5)
```

A.

```
10.0
```

B.

```
Base is: 4
```

C.

```
10.0
Base is: 4
```

[✓ **D.**

Nothing is displayed in the shell.]

Question 3: If you execute the code above as a program (with the), what will be displayed in the shell?

```python
def triangle_area(base, height):
    area = base * height / 2.0
    print("Area is:", area)
area = triangle_area(4, 5)

print(area)
```

A.

Python will report an error.

B.

```
Area is: 10.0
```

C.

```
Area is: 10.0
10.0
```

[✓ **D.**]

```
Area is: 10.0
None
```

Good coding style: code is not just for computers, it's also for humans!

- Is the code written clearly, efficiently, elegantly and logically?
- Is the code readable, e.g. good use of whitespace, effectively commented, meaningful variable names, uses language style conventions?
- Try to be `DRY` (**D**on't **R**epeat **Y**ourself): avoid duplicating code.
- In general, we will use `snake_case` 🐍 for variables and function names. Other naming conventions you may encounter in programming are `CamelCase` 🐪 and `drinkingCamelCase`.
- Use single-line comments to describe sections of code with a `#`
- Use triple-quotes to enclose multi-line "docstrings" which document your functions.
- Documentation can be viewed with `help(function_name)`.

```python
def triangle_area(base, height):
    """
    Computes the area of a triangle.

    Args:
        base: base length of the triangle
        height: height of the triangle

    Returns:
        Area of the triangle
    """
    return base * height / 2.0
```

Using constants in code: example computing `light_years`.

```python
SPEED_OF_LIGHT = 299792458
DAYS_PER_YEAR = 365.25
HOURS_PER_DAY = 24
SECONDS_PER_HOUR = 3600
SECONDS_PER_MINUTE = 60
MINUTES_PER_HOUR = 60

def light_minutes(distance):
   seconds = distance / SPEED_OF_LIGHT
   minutes = seconds / SECONDS_PER_MINUTE
   return minutes

def light_years(distance):
    minutes = light_minutes(distance)
    return minutes / (DAYS_PER_YEAR * HOURS_PER_DAY * MINUTES_PER_HOUR)
```

Define constants before any functions or expressions
(near the top of your scripts).

Let's revisit the `euclidean_distance` calculation from Lab 1 (now in a function).

First, write a docstring for this function!

```python
def euclidean_distance(x1, y1, x2, y2):
    """
    Calculates the euclidean distance between two points.

    Args:
        x1: x-coordinate of first point
        y1: y-coordinate of first point
        x2: x-coordinate of second point
        y2: y-coordinate of second point

    Returns:
        Distance between (x1, y1) and (x2, y2)
    """
    deltaX = x2 - x1
    deltaY = y2 - y1
    return (deltaX ** 2 + deltaY ** 2) ** (0.5)
```

What parts of this code are confusing? Can we make this code clearer?

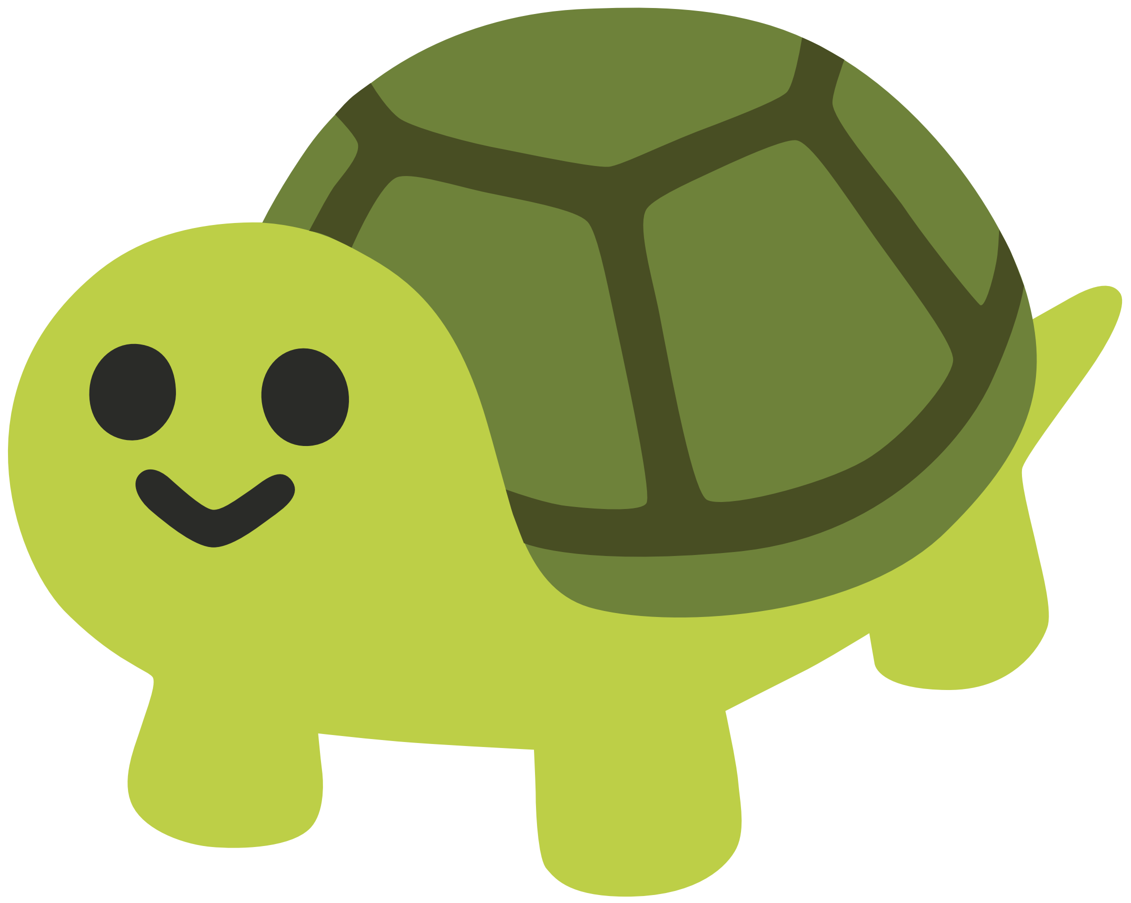
Modules allow us to keep a bunch of functions in a separate file and `import` what we need.

```python
# Import functions etc. with `math.` prefix
import math as math

# Shorthand for import math as math
import math

# Import all functions directly into current symbol table, i.e. with no prefix
from math import *

# Import specific function
from math import pow, sqrt
```

```python
import math

def euclidean_distance(x1, y1, x2, y2):
    deltaX = x2 - x1
    deltaY = y2 - y1
    return math.sqrt(math.pow(deltaX, 2) + math.pow(deltaY, 2))
```

Other modules to know about.

- `random`: with `randint(a, b)` which generates a random integer between `a` and `b` (inclusive). Also good to know about `seed` which is useful for "seeding" the random number generator.
- `turtle`: with functions like `forward`, `left`, `right` and more!



Example turtle program to draw a square.

```python
from turtle import *

def draw_square():
    forward(100)
    right(90)
    forward(100)
    right(90)
    forward(100)
    right(90)
    forward(100)
    right(90)

draw_square()
```

Hmm. This doesn't look very DRY. How can we make this better?

Summary

- Files are usually organized as:

```
# IMPORT MODULES

# DEFINE CONSTANTS

# DEFINE FUNCTIONS

# EVALUATE SOME EXPRESSIONS (if applicable)
```

- Use `#` to document specific (single) lines of code.
- Use `"""` to enclose multi-line docstrings to document functions.
- Ask yourself: is something being repeated (if so, DRY it up).
- So far, we have used `math`, `random` and `turtle` modules.
- Next time: loops!
- **Office hours:**
  Monday 10 - 11am, Tuesday 11am - 11:30am (my office, room 219) and Thursdays 2:30 - 4:30pm (room 224).

What I did this summer CS seminar (Part 2) on Friday at 12:30pm (with pizza)!

Summary

- Programs are usually organized as:

```
# IMPORT MODULES

# DEFINE CONSTANTS

# DEFINE FUNCTIONS

# EVALUATE SOME EXPRESSIONS (if applicable)
```

- Use `#` to document specific (single) lines of code.
- Use `"""` to enclose multi-line docstrings to document functions.
- Ask yourself: is something being repeated (if so, DRY it up).
- We have used `math`, `random` and `turtle` modules.
- Next time: loops!
- Come to office hours!
  Monday 10 - 11am, Tuesday 11am - 11:30am (my office, room 219) and Thursdays 2:30 - 4:30pm (room 224).

What I did this summer CS seminar (Part 2) on Friday at 12:30pm (with pizza)!