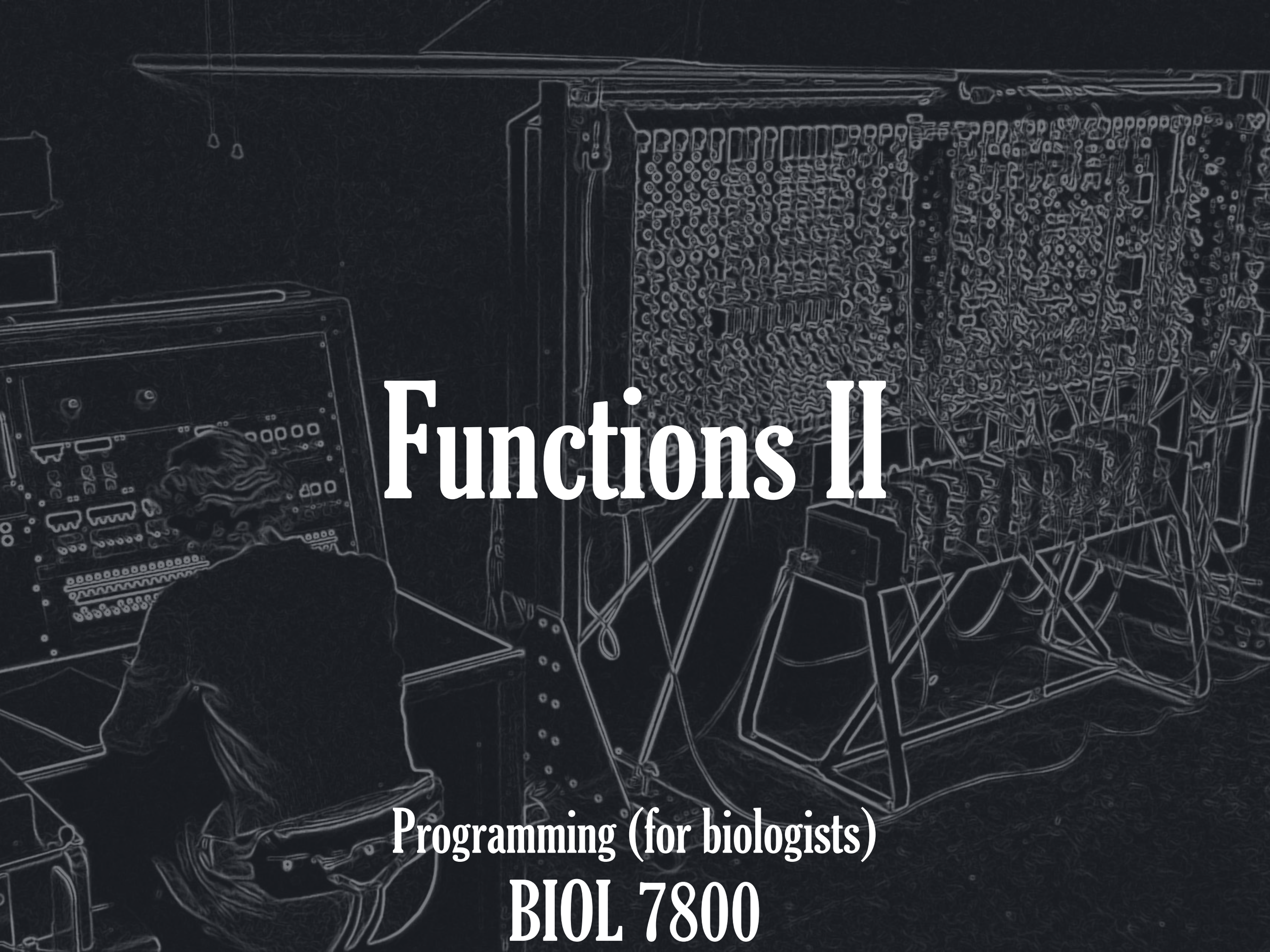


Functions II

Programming (for biologists)

BIOL 7800

Returns

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4   """
5   example.py
6
7   Created by Brant Faircloth on 3 February 2016
8   Copyright 2016 Brant C. Faircloth.  All rights reserved.
9   """
10
11
12  def function1(arg):
13      product = arg * arg
14      return product
15
16
17  def main():
18      function1(2)
19
20  if __name__ == '__main__':
21      main()
22
```

File 0 | Project 0 | No Issues | example.py* | 18:16 | LF | UTF-8 | Python

Returns

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
#!/usr/bin/env python
# encoding: utf-8

"""
example.py

Created by Brant Faircloth on 3 February 2016
Copyright 2016 Brant C. Faircloth.  All rights reserved.
"""


def function1(arg):
    product = arg * arg
    return product


def main():
    function1(2)

if __name__ == '__main__':
    main()

```

File 0 | Project 0 | No Issues | example.py* | 18:16 | LF | UTF-8 | Python

The main purpose of **return** is to return a result(s) that we can use later.

return allows us to divide our programs into atomic functions that are easier to **understand**, **debug**, and **use again**.

Returns

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
#!/usr/bin/env python
# encoding: utf-8

"""
example.py

Created by Brant Faircloth on 3 February 2016
Copyright 2016 Brant C. Faircloth.  All rights reserved.
"""


def function1(arg):
    product = arg * arg
    return product


def main():
    function1(2)

if __name__ == '__main__':
    main()

```

File 0 | Project 0 | No Issues | example.py* | 18:16 | LF | UTF-8 | Python

returns the result to **main()**

But what's wrong here?

Returns

No variable to “catch” the result

so, it “disappears”

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
2   # encoding: utf-8
3
4   """
5   example.py
6
7   Created by Brant Faircloth on 3 February 2016
8   Copyright 2016 Brant C. Faircloth.  All rights reserved.
9   """
10
11
12  def function1(arg):
13      product = arg * arg
14      return product
15
16
17  def main():
18      result = function1(2)
19      print result
20
21  if __name__ == '__main__':
22      main()
23
```

File 0 Project 0 ✔ No Issues example.py* 22:11 LF UTF-8 Python

Returns

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4   def function1(arg):
5       if arg <= 2:
6           product = arg * arg
7       else:
8           summ = arg + arg
9           return summ
10
11
12  def main():
13      result = function1(2)
14      print result
15
16  if __name__ == '__main__':
17      main()
18
```

What's wrong here?

Returns

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4   def function1(arg):
5       if arg <= 2:
6           product = arg * arg
7       else:
8           summ = arg + arg
9           return summ
10
11
12  def main():
13      result = function1(2)
14      print result
15
16  if __name__ == '__main__':
17      main()
18
```

Only returns if the **else** condition is True

so, sometimes we **return** a result

sometimes we do not

Returns

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4   def function1(arg):
5       if arg <= 2:
6           product = arg * arg
7       else:
8           summ = arg + arg
9       return summ
10
11
12  def main():
13      result = function1(2)
14      print result
15
16  if __name__ == '__main__':
17      main()
18
```

What's wrong here?

Returns

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4   def function1(arg):
5       if arg <= 2:
6           product = arg * arg
7       else:
8           summ = arg + arg
9       return summ
10
11
12  def main():
13      result = function1(2)
14      print result
15
16  if __name__ == '__main__':
17      main()
18
```

returns **summ**, butt…

Returns

So, how do we do this?

How do we correctly return **product** or **summ**?

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4   def function1(arg):
5       if arg <= 2:
6           product = arg * arg
7       else:
8           summ = arg + arg
9
10
11  def main():
12      result = function1(2)
13      print result
14
15  if __name__ == '__main__':
16      main()
17
```

Returns

Approach #1

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def function1(arg):
6       if arg <= 2:
7           result = arg * arg
8       else:
9           result = arg + arg
10      return result
11
12
13  def main():
14      result = function1(2)
15      print result
16
17  if __name__ == '__main__':
18      main()
19
```

} We assign both to **result**

← And we **return result**

File 0 | Project 0 | ✔ No Issues | example.py* | 4:1 | LF | UTF-8 | Python

Returns

Approach #2

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def function1(arg):
6       if arg <= 2:
7           product = arg * arg
8           return product
9       else:
10          summ = arg + arg
11          return summ
12
13
14  def main():
15      result = function1(2)
16      print result
17
18  if __name__ == '__main__':
19      main()
```

File 0 Project 0 No Issues example.py* 19:11 LF UTF-8 Python

But we can also use two **return** statements

Because both **return** statements are in an alternative conditional, only one of the two runs

Returns

using alternative conditionals

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def function1(arg):
6       if arg <= 2:
7           product = arg * arg
8           return product
9       else:
10          summ = arg + arg
11          return summ
12
13
14  def main():
15      result = function1(2)
16      print result
17
18  if __name__ == '__main__':
19      main()
```

File 0 Project 0 ✔ No Issues example.py* 19:11 LF UTF-8 Python

But we can also use two **return** statements

The function always **terminates** after it hits a **return**

Returns

using alternative conditionals

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def abs_value(x):
6       if x < 0:
7           return -x
8       if x > 0:
9           return x
10
11
12  def main():
13      result = abs_value(2)
14      print result
15
16  if __name__ == '__main__':
17      main()
18
```

File 0 | Project 0 | ✔ No Issues | example.py* | 18:1 | LF | UTF-8 | Python

When you use alternative conditionals to return a value you need to be careful!

What's wrong here?

Returns

using alternative conditionals

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def abs_value(x):
6       if x < 0:
7           return -x
8       if x > 0:
9           return x
10
11
12  def main():
13      result = abs_value(0)
14      print result
15
16  if __name__ == '__main__':
17      main()
18
```

File 0 Project 0 ✔ No Issues example.py* 18:1 LF UTF-8 Python

When you use alternative conditionals to return a value you need to be careful!

What's wrong here?

Returns

using alternative conditionals

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def abs_value(x):
6       if x < 0:
7           return -x
8       if x > 0:
9           return x
10
11
12  def main():
13      result = abs_value(0)
14      print result
15
16  if __name__ == '__main__':
17      main()
18
```

File 0 Project 0 ✔ No Issues example.py* 18:1 LF UTF-8 Python

When you use alternative conditionals to return a value you need to be careful!

What happens if we pass 0?

Returns

using alternative conditionals

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def abs_value(x):
6       if x < 0:
7           return -x
8       if x > 0:
9           return x
10
11
12  def main():
13      result = abs_value(0)
14      print result
15
16  if __name__ == '__main__':
17      main()
18
```

File 0 Project 0 ✔ No Issues exam

When you use conditionals you need to

What happens if we pass 0?

2. zsh

```
bcf at brant-4 in ~/Dropbox (faircloth-lab)/Classes/BIOL7800/temp
$ python example.py
None
```

Returns

terminate execution

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def function1(arg):
6       if arg <= 2:
7           product = arg * arg
8           return product
9       else:
10          summ = arg + arg
11          return summ
12      print("This will never run")
13
14
15  def main():
16      result = function1(2)
17      print result
18
19  if __name__ == '__main__':
20      main()
```

The function always **terminates** after it hits a **return**

Won't run.

File 0 | Project 0 | No Issues | example.py | 12:33 | LF | UTF-8 | Python

Returns

can be placed differently

Typical way you've been **return**ing values

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def function1(arg):
6       product = arg * arg
7       return product
8
9
10  def main():
11      result = function1(2)
12      print result
13
14  if __name__ == '__main__':
15      main()
16
```

variable is assigned to product

return product

File 0 | Project 0 | No Issues | example.py* | 7:5 | LF | UTF-8 | Python

Returns

can be placed differently

But we can also **return** the result of an expression

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def function1(arg):
6       return arg * arg
7
8
9   def main():
10      result = function1(2)
11      print result
12
13  if __name__ == '__main__':
14      main()
15
```

return result of expression

Which approach is “better”?

File 0 | Project 0 | No Issues | example.py* | 5:20 | LF | UTF-8 | Python

Boolean Functions

Functions can return **True** or **False**, just as they return other values

These are called **boolean functions!**

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def is_valid_dna(sequence):
6       if "N" or "X" in sequence:
7           return False
8       else:
9           return True
10
11
12  def main():
13      sequence = 'ACGTNNNNNGTAGCACTACG'
14      if is_valid_dna(sequence):
15          print("Hell yeah!")
16      else:
17          print("No. So sorry.")
18
19  if __name__ == '__main__':
20      main()
21
```

returns True or **False**

File 0 Project 0 ✔ No Issues example.py 17:20 LF UTF-8 Python

Boolean Functions

Functions can return **True** or **False**, just as they return other values

These are called **boolean functions!**

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def is_valid_dna(sequence):
6       if "N" or "X" in sequence:
7           return False
8       else:
9           return True
10
11
12  def main():
13      sequence = 'ACGTNNNNNGTAGCACTACG'
14      if is_valid_dna(sequence):
15          print("Hell yeah!")
16      else:
17          print("No. So sorry.")
18
19  if __name__ == '__main__':
20      main()
21
```

Common to name these as yes/no tests

return

File 0 | Project 0 | No Issues | example.py | 17:20 | LF | UTF-8 | Python

Boolean Functions

Functions can return **True** or **False**, just as they return other values

These are called **boolean functions!**

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def is_valid_dna(sequence):
6       if "N" or "X" in sequence:
7           return False
8       else:
9           return True
10
11
12  def main():
13      sequence = 'ACGTNNNNNGTAGCACTACG'
14      if is_valid_dna(sequence):
15          print("Hell yeah!")
16      else:
17          print("No. So sorry.")
18
19  if __name__ == '__main__':
20      main()
21
```

Common
these as

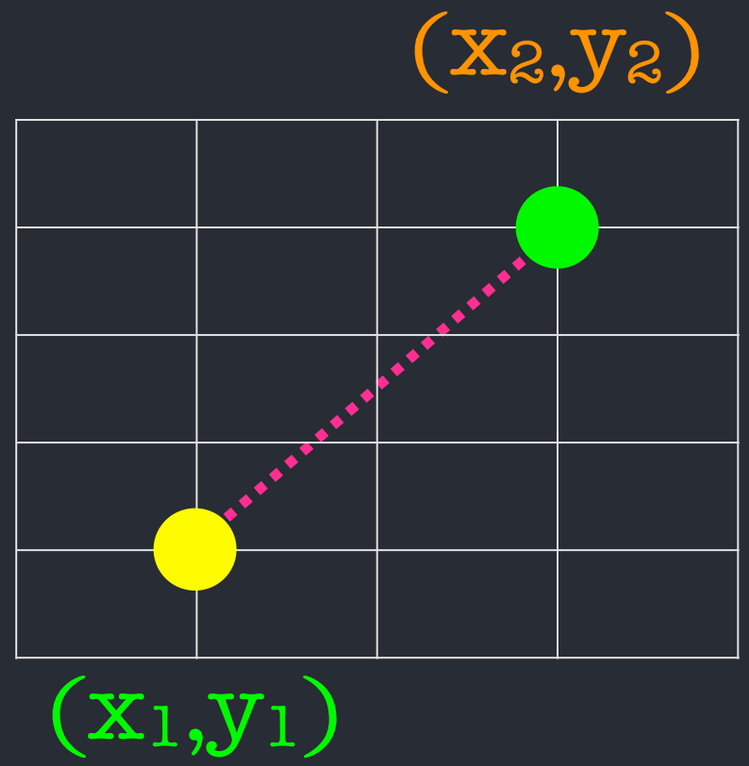
return

We can use these as **conditions**

File 0 | Project 0 | No Issues | example.py | 17:20 | LF | UTF-8 | Python

Incremental development

It's common to develop functions in a **step-by-step** process
testing the result as you **add complexity**



$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Incremental development

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def distance(x1, y1, x2, y2):
6       print("x1=", x1, "y1=", y1, "x2=", x2, "y2=", y2)
7
8
9   def main():
10      distance(1, 1, 3, 4)
11
12  if __name__ == '__main__':
13      main()
14
```

We want to break problem into component parts

And check each component, incrementally

So, first, we make sure variables are being passed correctly

Incremental development

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def distance(x1, y1, x2, y2):
6       delta_x = x2 - x1
7       delta_y = y2 - y1
8       print("delta_x: ", delta_x)
9       print("delta_y: ", delta_y)
10
11
12  def main():
13      distance(1, 1, 3, 4)
14
15  if __name__ == '__main__':
16      main()
17
```

} We compute **first** components of problem

} And check our results

If those are okay, move along….

Incremental development

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   def distance(x1, y1, x2, y2):
6       delta_x = x2 - x1
7       delta_y = y2 - y1
8       sum_squares = delta_x**2 + delta_y**2
9       print("sum of squares=", sum_squares)
10
11
12  def main():
13      distance(1, 1, 3, 4)
14
15  if __name__ == '__main__':
16      main()
17
```

We compute **second** component of problem

And check our results

Incremental development

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4   import math
5
6
7   def distance(x1, y1, x2, y2):
8       delta_x = x2 - x1
9       delta_y = y2 - y1
10      sum_squares = delta_x**2 + delta_y**2
11      distance = math.sqrt(sum_squares)
12      print("distance=", distance)
13
14
15  def main():
16      distance(1, 1, 3, 4)
17
18  if __name__ == '__main__':
19      main()
```

We compute **third** component of problem

And check our results

Incremental development

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
2   # encoding: utf-8
3
4   import math
5
6
7   def distance(x1, y1, x2, y2):
8       """compute distance between to cartesian coords"""
9       delta_x = x2 - x1
10      delta_y = y2 - y1
11      sum_squares = delta_x**2 + delta_y**2
12      distance = math.sqrt(sum_squares)
13      return distance
14
15
16  def main():
17      result = distance(1, 1, 3, 4)
18      print("The distance is", result)
19
20  if __name__ == '__main__':
21      main()
```

Add **description**

return result

print result