

Functions

Rodda John

06/22/2020

1 Clerical Matters

- Welcome!
- We're a large group:
 - Use Zoom well!
 - Chatting!
 - Raising hands!
 - Faster / slower!

1.1 Schedule

		SUNDAY June 21	MONDAY June 22	TUESDAY June 23	WEDNESDAY June 24	THURSDAY June 25	FRIDAY June 26	SATURDAY June 27
Morning Sessions	9		GROUP 1 9-11 AM Lesson 1: Basics Wrap-Up	GROUP 1 9-11 AM Lesson 2: Data Analysis	EVERYONE 9 AM - 12 PM Project Time	EVERYONE 9-11 AM Project Time	EVERYONE 9 AM - 12 PM Project Time	
	10							
	11		GROUP 2 11 AM - 1 PM Lesson 1: Basics Wrap-Up	GROUP 2 11 AM - 1 PM Lesson 2: Data Analysis		11-1 PM (each group will have a 15 minute meeting with Rodda)		
	12				12-1 PM		12-1 PM	
Break	1		1-2 PM	1-2 PM	GROUP 2 1-3 PM Lesson 4: Intro to AI	EVERYONE 1-5 PM Project Time	EVERYONE 1-2:30 PM Debrief Python	
Afternoon Sessions	2		EVERYONE 2-4 PM Project Time	GROUP 2 2-4 PM Lesson 3: Modeling				
	3				GROUP 1 3-5 PM Lesson 4: Intro to AI			
	4			GROUP 1 4-6 PM Lesson 3: Modeling		(all groups required to spend at least 2 hours working on project)		
	5							

1.2 Today

- Introduce you to writing your own functions
- Learn about breaking projects down into manageable pieces
- Start thinking algorithmically

2 Functions

- What is a function?
- A function is a procedure that execute code given certain inputs.
- It must have
 - A name
 - A definition (either local or in a library)
- It may have
 - Inputs (we will call these args, or arguments)
 - Outputs (we will call these return values)

2.1 Why are these useful?

- Why are functions the most useful programming tool?
- What are some use-cases?
- How can we conceptualize a function? How can we tell whether we ought make a function?

3 Functions in Python

- We need to learn how to do two things:
 - Calling functions (executing them, using them)
 - Defining functions (making our own)

3.1 Calling Functions

```
print('Hello World!')
```

```
len([1, 2, 3, 4, 5, 6])
```

```
input()
```

- How many **args** does each function have?
- What is each functions output?

3.2 Defining Functions

3.2.1 Simple Function

- This function:
 - Accepts two numerical inputs
 - Returns the product

```
def product(x, y):  
    return x * y
```

3.2.2 Sum of a list

- This function:
 - Accepts a list
 - Returns the sum of the elements of the list

```
def sum_of_list(l):  
    to_return = 0 # Tracks the sum of the list  
  
    i = 0 # Iterator to iterate through the list  
  
    while i < len(l):  
to_return += l[i]  
    i = i + 1  
  
    return to_return
```

3.2.3 Generally

- `def` opens a function definition block
- A name is required, as well as `()` to denote the args
- Take note of the `:`
- As with `if` statements, and `while` loops, the code in the function is indented

3.2.4 Thus

```
def <name>(<args>):  
--> code  
--> (optional) return <to_return>
```

4 Let's Try it Out

4.1 First Problem

- Write a function (`sum`) that:
 - Accepts two numerical inputs
 - Returns the sum
- How can we test this function?
 - What are the edge cases?

4.1.1 First Problem Solution

```
def sum(x, y):  
    return x + y
```

```
sum(1, 2)    # Should be 3  
sum(4, -2)   # Should be 2  
sum(4, 0)    # Should be 4
```

4.2 Second Problem

- Write a function (`product`) that:
 - Accepts two numerical inputs
 - Returns the product
 - Does not use the `*` operator, and instead uses the `sum` function we defined above

4.2.1 Second Problem Solution

```
def product(x, y):  
    to_return = 0
```

```

    iterator = 0

    while iterator < y:
to_return = to_return + x

    iterator = iterator + 1

    return to_return

product(1, 2)    # Should be 2
product(100, 2) # Should be 200
product(2, 100) # Should be 200
product(2, .5)  # Should be 1

```

4.3 Third Problem

- Write a function(`divisible_by`) that:
 - Accepts two numerical inputs (`number`, and `factor`)
 - Returns true if and only if `number` is evenly divisible by `factor`
 - Recall the `%` operator, which returns the remainder of the first operand by the second

4.3.1 Third Problem Solution

```

def divisible_by(number, factor):
    remainder = number % factor

    if remainder == 0:
return True
    else:
return False

divisible_by(4, 2) # Should be True
divisible_by(4, 1) # Should be True
divisible_by(15, 3) # Should be True
divisible_by(15, 4) # Should be False

```

4.3.2 Or, even shorter

```

def divisible_by(number, factor):

```

```
    remainder = number % factor

    if remainder == 0:
return True
    return False
```

4.3.3 Or, even shorter

```
def divisible_by(number, factor):
    return number % factor == 0
```