



CoGrammar

Programming with Functions (Part Two)

**SKILLS
FOR LIFE**

SKILLS BOOTCAMPS



Department
for Education

Data Science Lecture Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
(FBV: Mutual Respect.)
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Open Classes.
You can submit these questions here: [Open Class Questions](#)

Data Science Lecture Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support
- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: [Feedback on Lectures](#)

Lecture Objectives

- A recap on using Python modules and creating our own functions.
- Learn how to harness the power of Python Virtual Environments.

A recap on Functions

- ★ It is a reusable and organised block of code.
- ★ Sometimes it's called, a 'method'.
- ★ Similar to functions in maths - $f(x)$ takes an input of x and produces an output.
- ★ Also useful for abstraction.
 - Abstraction - the concept of defining complex functionality by using a single term. Great for defining high-level bits of functionality.

A recap on Functions

- ★ **Reusable code** - Sometimes we'll need to do the same thing multiple times.
- ★ **Error checking / validation** - Makes this process easier, as the logic is placed in one place that is easy to find.
- ★ **Dividing code up into manageable chunks** - Makes our code easier to read and understand.
- ★ **Rapid development** - The same functionality does not need to be defined again.

Lambda Functions

- ★ With Python, we can create a special type of function. Called lambda functions.
- ★ They are special because, it's a small anonymous function, that only has one expression.

```
lambda : print('Hello World!')  
# This is a simple lambda function that prints hello world.  
  
# to get output, do the following :  
greeting = lambda : print('Why hello there')  
greeting() # Output : Why hello there
```

Lambda Functions

- ★ We call it an anonymous functions because it doesn't have a name.
- ★ You may have also noticed that to define a lambda function, we use the keyword : lambda, instead of the traditional def keyword to create functions.
- ★ It's very useful for creating code to do small tasks, by creating less code for it.
- ★ Additionally lambda functions are called immediately when created.

Lambda Examples

Lambda function that adds 2 to any number

```
add_two = lambda x : x + 2
```

```
print(add_two(10)) # Output : 12
```

Lambda function that will return the cubed version of any number

```
cube = lambda y : y ** 3
```

```
print(cube(7)) # Output : 343
```

Lambda function that creates a custom greeting.

```
greeting = lambda name : print(f'Welcome back {name}')
```

```
greeting("Johnathan")
```

Recursive Functions

- ★ We can define recursion as a process where we define something in terms of itself.
- ★ In Python, this means we can call a function within itself, making it a recursive function.
- ★ This can be advantageous in terms of breaking down a problem into its simplest form as well as making your solution look clean and efficient.

Recursion Example

```
# A recursive function to acquire the factorial of a value.
def factorial(n):
    # two simple steps to creating a recursive function :
    # Step One - find the base case, the simplest version of our problem
    # Step Two - work towards the base case, which will end our recursion.
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5)) # Output : 5 x 4 x 3 x 2 x 1 = 120
```

More on Recursion

- ★ **There are a few drawbacks when it comes to using recursion. Such as :**
 - **It does take a lot of memory when running, which makes it computationally expensive.**
 - **The logic placed within the function can be difficult to look at and debug should there be issues.**

Let's Breathe

Let's take a small break before moving on to the next topic.

Virtual Environments

It is highly recommended to use virtual environments in python for all our projects, reason being is that we can come across issues with our external packages which virtual environments can mitigate.

It is easy to install and activate to allow us to install pip packages for our projects and make our projects look and work a bit cleaner.

CoGrammar

Q & A SECTION

**Please use this time to ask
any questions relating to the
topic, should you have any.**



CoGrammar

Thank you for joining!