Functions

Processing data again and again!

Functions are reuseable chunks of code, that allow us to repeat processes.
They usually process data in some way - but they don't have to!
Functions can really make your code work for you.

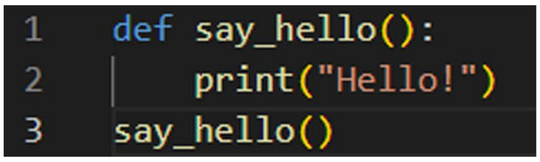
Function Syntax

The write way to function

There are already functions built-in to Python, and we use them regularly. We can write our own, though!
We we create a new function, we begin by defining it - so we say **def** - without this key word, Python wouldn't know we were creating a new function, and it would be looking for a pre-exisiting function instead!
Then we say the function name - this can be anything! You should use snake_case, and make your function name easy to type and relevant to the action of the function!
After that, we need a set of brackets (). They're the rules! As we're still defining the function, we pop a colon : at the end too. When we go on to a new line we indent, and anything indented underneath is part of what happens when we run the function!

Making It Work

We can define our functions to do pretty much anything we want! The function below's job is to say hello!



```
1   def say_hello():
2       print("Hello!")
3   say_hello()
```

Once we've written our instructions out we need to actually make the function run! We do this by calling it - using its name. Don't forget the brackets! In the example above, we do this on line 3. Note how it's no longer indented?

Once we've made the function, we can call it anywhere after, as many times as we like!

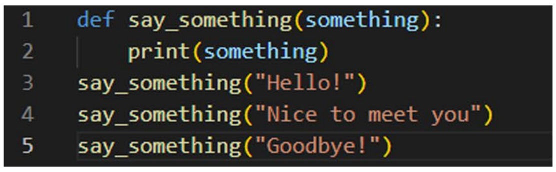
But what are the brackets for?!?!

(That's a very good question)

Functions can allow us to process data. We can define what we want to do to the data when we write out the instructions for the function - but how do we let it know what the data is?

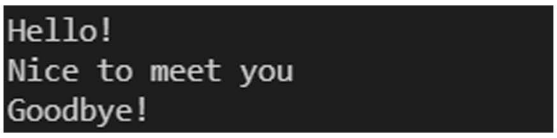
If we want our function to work with some data, we should let it know when we're defining it. We can create a variable name we want to reference the data by, and put that between the brackets. This is called a **parameter**. You can put as many **parameters** as you like, even none!

Inside the function, we'll write out the instructions for what we want to do with the data - in this case we'll just print it - but we can do anything!.



```
def say_something(something):
    print(something)
say_something("Hello!")
say_something("Nice to meet you")
say_something("Goodbye!")
```

terminal response:



```
Hello!
Nice to meet you
Goodbye!
```

When we call the function, we have to give that **parameter** a value, so we enter that between the brackets. This is called an **arguement**. In our example, **something** is the parameter, **"Hello", "Nice to meet you"**, and **"Goodbye"** are the **arguements**. Each time we call the function now, we'll need to enter as many **arguements** as there are **parameters** - but the value of the **arguement** will be different every time, allowing us perform the same process on different data!

How is this helpful?

Think back to examples we've looked at in class - like a cash machine. When you go to withdraw money, a function will have already been written to let you withdraw cash, and called every time you select "withdraw." Think about any parameters it might have like "amount". For every person, the value of amount will be different, but the process will be the same!

{cn}®