Functions Examples using Keyword arguments

What are we going to do?

We're going to take a look at how to use keyword arguments in functions. We're also going to take a look at how to pass in "nonspecific" arguments to a function (and what the heck that is).

- how to use the `**kwargs` keyword to pass a dictionary of arguments to a function.
- how to use the `*args` keyword to pass a list of arguments to a function.

Why is this important?

This is going to be a technique that you might not define every day but it's something that you'll see when using packages (other people's code) in the future.

Steps

1. Let's create a function called `pizza.py` and make a function named `build_pizza` that's going to take in the size and crust.

- We're going to add toppings a little later.

```
def make_pizza(size, crust):
    """Summarize the pizza we are about to make."""
    print("\nMaking a " + str(size) + "-inch pizza with " +
          crust + " crust.")

# Let's call this function
if __name__ == "__main__":
    make_pizza(16, 'thin')
    make_pizza(12, 'thick')
    make_pizza(18, 'medium')
```

- The output should look like this:

```
$ python pizza.py

Making a 16-inch pizza with thin crust.

Making a 12-inch pizza with thick crust.

Making a 18-inch pizza with medium crust.
```

- In the above `make_pizza` function you see that we to need call the function with the arguments in the correct order because we're using positional arguments.

- What if we want to change the order of the arguments? Or what if we want to add a bit more information to the function call?

2. Let's change these to use keyword arguments.
    - this allows us to use the arguments in any order and also add a bit more information to the function call to make it a bit more obvious.
    - Let's change this!

```
# Note below here we're adding a default of None to the arguments, which
makes it a keyword argument with a default value of None.
def make_pizza(size=None, crust=None):
    """Summarize the pizza we are about to make."""
    print("\nMaking a " + str(size) + "-inch pizza with " +
          crust + " crust.")

# ... function calls ...
```

- Now let's change the order of the arguments and change the function calls to use the keyword arguments rather than the positional arguments.

```
# ... function definition ...
if __name__ == "__main__":
    make_pizza(size=16, crust='thin')
    make_pizza(crust='thick', size=12)
    make_pizza(size=18, crust='medium')
```

- Note the output is the same as before.
- But note that even if we changed the order of the arguments, the output is still the same!

3. Let's add toppings to the pizza and talk about nonspecific arguments.

- Now when we're making a pizza we can have any number of toppings.
    - You can have a "cheese" pizza, or a "pepperoni" and "mushroom" pizza, or a "pineapple", "ham", and "ricotta" pizza where all of these pizzas are valid.
    - We're going to modify our `make_pizza` to take in a nonspecific number of toppings. This is done with the `*args` keyword. Let's take a look at how this works.

```
# Note below we're adding *args to the function definition.
# this allows us to pass in a nonspecific number of arguments.
def make_pizza(size=None, crust=None, *args):
    """Summarize the pizza we are about to make."""
    print("\nMaking a " + str(size) + "-inch pizza with " +
          crust + " crust.")
    print("The Toppings added are:")
    # note here that toppings is a tuple that you can loop through
    # you can also insert a breakpoint here to see exactly what it is.
    for topping in args:
```

```
        print("- " + topping)

# ... function calls ...
```

- Now let's call this function with a nonspecific number of arguments.
- Note the order here of arguments and keyword arguments is always:
    - positional arguments
    - nonspecific position arguments (*args)
    - keyword arguments
    - nonspecific keyword arguments (**kwargs)

```
# ... function definition ...
if __name__ == "__main__":
    make_pizza('cheese', size=16, crust='thin')
    make_pizza('pepperoni', 'mushroom', crust='thick', size=12)
    make_pizza('pineapple', 'ham', 'ricotta', size=18, crust='medium')
```

- You can see above that we're passing a nonspecific number of arguments to the function, but we're also defining the type of crust and the size of the pizza.
- The output of the function should look like this:

```
$ python pizza.py

Making a 16-inch pizza with thin crust.
The Toppings added are:
- cheese

Making a 12-inch pizza with thick crust.
The Toppings added are:
- pepperoni
- mushroom

Making a 18-inch pizza with medium crust.
The Toppings added are:
- pineapple
- ham
- ricotta
```

4. Some people might have special requests to their pizza. Let's add a nonspecific number of keyword arguments (**kwargs) to our function to handle this.

- let's use the **kwargs keyword to pass a dictionary of arguments to a function.

```
def make_pizza(*args, size=None, crust=None, **kwargs):
    """Summarize the pizza we are about to make."""
    # ... code removed for brevity ...
```

```
    if kwargs:
        print("Special instructions for the pizza are:")
        for key, value in kwargs.items():
            print(F"- {key}: {value}")
```

- you can see here that the `kwargs` is a dictionary that you can use in your function.
- Let's call this function with a nonspecific number of arguments and keyword arguments.

```
# ... function definition ...


if __name__ == "__main__":
    make_pizza('cheese', size=16, crust='thin', cheese="double")
    make_pizza('pepperoni', 'mushroom', crust='thick', size=12)
    make_pizza('pineapple', 'ham', 'ricotta', size=18, crust='medium',
ham="extra", ricotta="extra")
```

- observe how we're passing different keyword arguments to the function.
    - in the first function call, we're adding `cheese="double"` to the keyword arguments.
    - in the second function call, we don't add any extra keyword arguments.
    - in the third function call, we're adding `ham="extra"` and `ricotta="extra"` to the keyword arguments.
- The output should look like below.

```
$ python pizza.py

Making a 16-inch pizza with thin crust.
The Toppings added are:
- cheese
Special instructions for the pizza are:
- cheese: double

Making a 12-inch pizza with thick crust.
The Toppings added are:
- pepperoni
- mushroom

Making a 18-inch pizza with medium crust.
The Toppings added are:
- pineapple
- ham
- ricotta
Special instructions for the pizza are:
- ham: extra
- ricotta: extra
```

5. Let's make an order builder here and import functions from the `pizza.py` module.

- remove the function calls from the `pizza.py` file (anything in under the scope of `if __name__ == "__main__":`).
- let's create a file a file named `__init__.py` in our current folder.
    - this file is required for python to recognize this folder as a module (we'll need this later on when we organize our code into different folders).
- let's create a file named `order_builder.py` and get the details from the user.

```
if __name__ == "__main__":
    ordering_pizza = True
    while ordering_pizza:
        # see if the user is done ordering pizzas
        ordering = input("Are you done ordering pizzas? (y/n): ")
        if ordering == "n":
            ordering_pizza = False
            continue
        # get the size and crust
        size = int(input("What size pizza would you like? (12, 16, 18):
"))
        crust = input("What type of crust would you like? (thin, medium,
thick): ")
        # get the toppings
        toppings = []
        topping = input("What topping would you like? (enter 'done' when
finished): ")
        while topping != "done":
            toppings.append(topping)
            topping = input("What topping would you like? (enter 'done'
when finished): ")

        # get the special requests.
        special_requests = {}
        special_request_topping = input("What modification to toppings?
(enter 'done' when finished): ")
        while special_request_topping != "done":
            special_request_amount = input("How much? (light, extra,
double): ")
            special_requests[special_request_topping] =
special_request_amount
            special_request_topping = input("What modification to
toppings? (enter 'done' when finished): ")
```

- Let's talk about the pieces that we're getting from the user.
    - we're getting the size and crust of the pizza.
    - we're getting the toppings of the pizza as a list of toppings
    - we're getting the special requests as a dictionary of toppings and the amount of the topping.

6. Let's import the `make_pizza` function from the `pizza.py` module and call it with the arguments that we got from the user.

```
from pizza import make_pizza

if __name__ == "__main__":
    ordering_pizza = True
    while ordering_pizza:
        # ... code removed for brevity ...

        make_pizza(*toppings, size=size, crust=crust, **special_requests)
```

- Note that we're using the * and ** to unpack the list and dictionary into the function call.
    - this is because the `make_pizza` function is expecting a nonspecific number of arguments and keyword arguments.
    - so if you want to pass a list of arguments to a function that's expecting a nonspecific number of arguments. You can unpack the list with * and pass it to the function.
    - if you want to pass a dictionary of arguments to a function that's expecting a nonspecific number of keyword arguments. You can unpack the dictionary with ** and pass it to the function.
- The output should look like this:

```
Are you done ordering pizzas? (y/n): y
What size pizza would you like? (12, 16, 18): 12
What type of crust would you like? (thin, medium, thick): medium
What topping would you like? (enter 'done' when finished): cheese
What topping would you like? (enter 'done' when finished): pepperoni
What topping would you like? (enter 'done' when finished): done
What modification to toppings? (enter 'done' when finished): cheese
How much? (light, extra, double): extra
What modification to toppings? (enter 'done' when finished): done

Making a 12-inch pizza with medium crust.
Toppings added are:
- cheese
- pepperoni
Special instructions for the pizza are:
- cheese: extra
Are you done ordering pizzas? (y/n): n
```

7. Let's talk about the different ways that you can import code from modules. You can import the module in different ways.

- you can import the entire module

```
import pizza
# usage in the file.
pizza.make_pizza('cheese', size=16, crust='thin', cheese="double")
```

- you can import all of the functions in the module

```
from pizza import *
# usage in the file.
make_pizza('cheese', size=16, crust='thin', cheese="double")
```

- you can import a specific function from the module

```
from pizza import make_pizza
# usage in the file.
make_pizza('cheese', size=16, crust='thin', cheese="double")
```

- you can import a specific function from the module and rename it

```
from pizza import make_pizza as mp
# usage in the file.
mp('cheese', size=16, crust='thin', cheese="double")
```

- you can import a module and give it a nickname

```
import pizza as p
# usage in the file.
p.make_pizza('cheese', size=16, crust='thin', cheese="double")
```

What did we learn?

- How to use the `**kwargs` keyword to pass a dictionary of arguments to a function.
- How to use the `*args` keyword to pass a list of arguments to a function.
- How to import functions from a module and all of the different ways that you can import functions from a module.