Python Class Notes

Clarusway

Functions, Arguments, Modules

Nice to have VSCode Extentions:

- Jupyter

Needs

- Python (for Windows OS: add to the path while installing!)

Summary

- Functions
  - `print` vs `return`
- Arguments
  - Arbitrary Arguments, *args
  - Keyword Arguments
  - Arbitrary Keyword Arguments, **kwargs
  - Ordering Arguments in a Function
  - Default Parameter Value
- Modules
  - Creating modules
  - Creating modules in a seperate directory
  - Standard Modules

Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Naming functions varies, but in django you will genarally see snake_case.

```
# In Python a function is defined using the def keyword:
def my_function():
  return "Hello from a function"

# To call a function, use the function name followed by parenthesis:
my_function()
```

print vs return

`print` and `return` are two different ways to output data from a function in Python.

`print` is a built-in Python function that outputs text to the console. When you use print inside a function, the text is printed to the console when the function is called. However, print does not return a value from the function. That means that if you try to assign the result of a print statement to a variable or use it in an expression, you'll get None.

Here's an example of a function that uses print to output a message:

```
def greet(name):
    print(f"Hello, {name}!")

greet("Alice") # outputs `Hello Alice`; but,

result = greet("Alice")
print(result) # outputs None
```

When you call greet("Alice"), the function outputs Hello, Alice! to the console. However, if you try to assign the result of greet("Alice") to a variable, like result = greet("Alice"), result will be None.

On the other hand, `return` is a keyword in Python that lets you return a value from a function. When you use return inside a function, the function stops executing and returns the value to the code that called the function. That means you can use the result of a function call in an expression or assign it to a variable.

Here's an example of a function that uses return to return a value:

```
def add(a, b):
    return a + b
```

```
result = add(3, 4)
print(result) # outputs 7
```

When you call add(3, 4), the function returns 7. You can then assign that value to a variable, like result = add(3, 4), and use it in an expression or output it to the console.

In summary, print outputs text to the console, while return returns a value from a function that can be used in an expression or assigned to a variable.

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses.

You can add as many arguments as you want, just separate them with a comma.

```
# The following example has a function with one argument (name). When the function
is called, we pass along a name, which is used inside the function to print the
full name.
def my_function(name):
    retrun f"Hello {name} how are you?"

my_function("John")


# This function expects 2 arguments, and gets 2 arguments:
def my_function(name, last_name):
    retrun f"Hello {name} {last_name} how are you?"

my_function("John", "Smith")
```

Arbitrary Arguments, *args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

If the number of arguments is unknown, add a * before the parameter name:

```
def my_function(*kids):
  return "The youngest child is " + kids[0]

my_function("Hans", "Angel", "Lilly")
```

Keyword Arguments

You can also send arguments with the key = value syntax.

```
# This way the order of the arguments does not matter.
def my_function(child3, child2, child1):
  return "The youngest child is " + child3

my_function(child1 = "Hans", child2 = "Angel", child3 = "Lilly")
```

Arbitrary Keyword Arguments, **kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly:

```
# If the number of keyword arguments is unknown, add a double ** before the
parameter name:

def my_function(**kids):
    for kid in kids.values():
        print(f"Hello {kid}")

my_function(kid1 = "John", kid2 = "Lilly")
```

Ordering Arguments in a Function

The correct order for your parameters is:

- Standard arguments
- *args arguments
- **kwargs arguments

```
# correct_function_definition.py
def my_function(a, b, *args, **kwargs):
    pass
```

Default Parameter Value

If we call the function without argument, it uses the default value:

```
def my_function(country = "Norway"):
  return "I am from " + country

my_function("Sweden")
my_function()
```

Modules

In Python, a module is a file that contains Python code, usually with a specific functionality or purpose. Modules allow you to organize your code into separate files, making it easier to manage and reuse. When you want to use the functionality of a module in your code, you simply import it using the import statement.

Creating modules

To create a module, you simply create a Python file with the `.py` extension and define your functions or classes in it. You can then import the module into other Python files and use its functions and classes as needed.

```
# example_module.py

def add_numbers(x, y):
    return x + y

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

You can then import the module in another Python file and use its functions like this:

```
import example_module

result = example_module.add_numbers(3, 4)
```

In this example, we import the example_module module and call its add_numbers() function to add 3 and 4 together.

Creating modules in a seperate directory

The `__init__.py` files are required to make Python treat directories containing the file as packages.

In Python, an `__init__.py` file is a special file that is used to mark a directory as a Python package. When Python imports a package, it looks for an `__init__.py` file in the package's directory and runs any code inside it.

If the `__init__.py` file is empty, it simply marks the directory as a package and doesn't run any code. `

By including this file in a directory, you're indicating to Python that the directory should be treated as a package and that any Python modules or subpackages inside it can be imported using the import statement.`

An example folder structure stated in the documenation is:

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
```

```
      formats/                Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
      effects/                Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
      filters/                Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
```

Standard Modules

Standard modules in Python are pre-built libraries that come with the Python programming language. These modules provide a wide range of functionalities that can be used in various types of applications. Some examples of standard modules in Python include:

- `math` provides mathematical functions like trigonometric, logarithmic, and exponential functions,
- `datetime` provides classes for working with dates and times,
- `random` provides functions for generating random numbers and making random choices.

```python
# Examples:
import random

# Generate a random integer between 1 and 10 (inclusive)
random_int = random.randint(1, 10)
print(random_int)

# or;
from datetime import datetime

# Create a datetime object for the current date and time
now = datetime.now()
print(now)

# or;
import math

# using math module functions
x = math.sqrt(25)
print(x)  # 5.0
```

Standard modules are included with every installation of Python, and they can be imported and used in any Python program. By using standard modules, developers can save time and effort by reusing existing code and functionality rather than building everything from scratch.

☺ **Happy Coding!** ✍

Clarusway