Declaration on Plagiarism

Assignment Submission Form

This form must be filled in and completed by the student(s) submitting an assignment

Name(s): **Andrew Finn**
Student Number: [redacted]
Programme: **CASE3**
Module Code: **CA341**
Assignment Title: **Report: Analytical Analysis of a Programming Language: Python**
Submission Date: **December 2020**
Module Coordinator: **Dr. David Sinclair**

Name: **Andrew Finn** Date: **Dec 2020**

Table Of Contents

Analytical Analysis of a Programming Language: Python

Andrew Finn - 18402034

Introduction

What is Python? “Python is an interpreted, interactive, object-oriented, high level programming language.”[1] Python is also a multi-paradigm programming language.

Structure

Indentation

Python, unlike the majority other programming languages, utilises whitespace for indentation rather than utilizing curly brackets and keywords which is the standard for most other programming languages. This method of indentation is also known as the off-side rule[2].

```python
def is_odd(number):
    if number % 2 != 0:
        return True
    else:
        return False
```

I personally feel this method of delimiting blocks of code make for more readable code as well as making the flow of the code easier to follow. This use of off-side intention is one of Python’s love it or hate it features. However I personally prefer this method of indention for the above reasons, however I am biased due to learning Python as my first programming language and as such am more familiar to this style of programming.

Interpreted

Python is an interpreted programming language, this means python code can not be run directly; it instead must be passed to an ‘interpreter’. The interpreter will then read the source code and execute the program, therefore there is only one step from source code to execution. This is compared to compiled languages' two step process where code is passed through a compiler that creates executable machine code that can then be executed.

This interpreted approach is significantly slower than the compiled approach as the interpreter must generate machine code on a line by line basis as the program is interpreted. However the advantage is interoperability, a developer need only provide source code to the intended system which the interpreter can then execute. This means that only one ‘package’ needs to be distributed. Compiled programs in comparison are compiled for the system they are compiled on, meaning multiple executables are needed for interoperability. Python files are also significantly smaller than their compiled counterparts as only the source code is transferred compared to a binary file containing everything needed to run the program.

With an interrupted language code debugging is also more difficult due to the fact code debugging occurs at run-time.

```python
from sys import platform


def is_linux():
    if platform == "linux" or platform == "linux2":
        return True
    else:
        # “var_that_dosnt_exist” was never defined
        var_that_dosnt_exist += 1
        return False
```

For example in the code above if a developer was testing the program using a Linux operating system they would never catch the NameError that would occur on non Linux systems when trying to perform addition to an undefined variable. In a compiled language this error would have been caught by the compiler and would have been easily found and fixed.

High level language

Dynamically typed

```python
s = "Hello"
s = 123
s = [1, 2, 3]
```

The above code is valid Python code. For those more familiar with statically typed languages such as Java the above extract will be completely forign. Python is able to infer data types at assignment (or reassignment), i.e. Python is automatically able to assign the type of a variable such as an integer string… etc. by looking at the given data.

Also you will note the ability to reassign the variable to different data types from strings, to integers, to an array. This is because Python is not a Type-safe language. This lack of type safety can lead to elegant solutions but can also have unintended consequences, for example:

```python
def is_even(number):
    if number % 2 == 0:
        return True
    else:
        return False
```

The above code will return a Boolean true or false depending on if a given Integer is True or False. This seems straightforward but can lead to unforeseen consequences, for example if a string is passed to the above function the interpreter will raise an exception as “number” which represents a string will have no modulo operation. As such developers need to be careful not to accidentally reassign variables.
It is also important to note that a function does not a predefined output, for example:

```python
def sample(number):
    if number == 1:
        return True
    elif number == 2:
        return "You passed the number 2"
    elif number == 3:
        return [3]
    else:
        return 1.6
```

The above code is valid python code, depending on the input the function will return a different output type. This can be useful for example to return False if a solution is not possible in a mathematical function but when utilizing this functionality careful consideration to avoid runtime error as a result of treating the output as a singular desired type.

Programming paradigms

"Jack of all trades, master of none!"
The above is a fair comment on Python, it gives the developer the ability to programming a variety of programming paradigms. Such As:

Functional

Python can be used to write in a functional style, providing the developer adheres to certain rules,

Functions

For Python to be used as a functional language function must be deterministic, this means that the function must return the same output given the same input, as well as returning the same type regardless of input. Functions in the functional paradigm must also not modify external data and instead return a new object. For example:

```python
def is_even_list(input_list):
    output_list = []

    for number in input_list:
        if number % 2 == 0:
            output_list.append(True)
        else:
            output_list.append(False)

    return output_list
```

The above code will always return a list, the list will always be the same size as the input list and the list will be made up of booleans. This function is therefore in the function paradigm and deterministic.

Recursion

Functional Python also allows recursion, which when used correctly can create powerful and elegant solutions, the best known example of this is the recursive solution for the Fibonacci sequence.

```python
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

The above is an example of how recursion can make a problem that at first glance may be hard to translate in code but utilizing recursion can provide a straightforward and easily followed solution.

Immutability

Python also provides immutable data types such as Strings and Tuples which are a key requirement of functional programming. The presence and usage of immutable data types can reduce errors as the values can not be modified, however these data types can be appended to. For example:

```python
immutable_expample = ("a",  "b",  "c")
immutable_expample[-1] = "d"
# TypeError: 'tuple' object does not support item assignment
immutable_expample.append("e")
>> ("a", "b", "c" , "e")
```

The above code shows that tuples can not be modified and that tuples in python support the append operation.

Lambda & Comprehensions

Of my personal favourite features in Python is the presence of list comprehensions and lambda expressions. For example here a standard solution for the FizzBuzz problem:

```
def fizzbuzz(n):
    output_list = []
    for i in range(1, n):
        if i % 3 == 0 and i % 5 == 0:
            output_list.append("FizzBuzz")
        elif i % 3 == 0:
            output_list.append("Fizz")
        elif i % 5 == 0:
            output_list.append("Buzz")
        else:
            output_list.append(i)
    return output_list
```

And now here a solution utilising list comprehensions:

```
def fizzbuzz(n):
    return ["FizzBuzz" if (i % 3 == 0 and i % 5 == 0)
            else "Fizz" if (i % 3 == 0)
            else "Buzz" if (i % 5 == 0) else i for i in range(1, n)]
```

The ability to use these list comprehensions allows for complex problems to be solved in one line but still be very readable and understandable.

The presence of lambda calculus also opens up a plethora of options when creating programs for example:

```
def lambda_example(n):
    return lambda a: a * n


double = lambda_example(2)
triple = lambda_example(3)

double(5)
>> 10
triple(5)
>> 15
```

The usage of lambda calculus in the above example removed the need for repetition of functions and can provide a building block for complex functions. Lambda can also facilitate the creation of anonymous functions.

Object Oriented Programming

Python can also be used to program in the object oriented paradigm as since its creation has object oriented features, Python's OOP features are a descendant of C++ OOP features.

Classes & Objects

Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name.

Python classes can be defined as such:

```
class Car:
    def __init__(self, reg_plate, make, model):
        self.reg_plate = reg_plate
        self.make = make
        self.model = model
```

The above is an example of a class in Python, that implements a car class. This class has no methods and as such in its current state is very similar in purpose to structures/structs in other languages.

```
 car1 = Car("141-D-1234", "Ford", "Transit")
```

The above is an example of an initialisation of the Car class. "car1" is now known as an object as it is an instance of the Car class.
The real power of these classes comes when you add functions/methods to classes. For example,

```
reg_counties = {"D": "Dublin", …, "C": "Cork"}


class Vehicle:
    def __init__(self, reg_plate, make, model):
        self.reg_plate = reg_plate
        self.make = make
        self.model = model

    def get_county(self):
        l = self.reg_plate.split("-")
        return reg_counties[l[1]]
```

Now in the given example you can call car1.get_county and be returned “Dublin”. The same will be true for van1.get_county() as this method is inherited as van and car are children of the vehicle class

Inheritance

These classes can also be extended I.E. they can used as a building block for other classes for example:

```
reg_counties = {"D": "Dublin", ..., "C": "Cork"}


class Vehicle:
    def __init__(self, reg_plate, make, model):
        self.reg_plate = reg_plate
        self.make = make
        self.model = model

    def get_county(self):
        l = self.reg_plate.split("-")
        return reg_counties[l[1]]


class Car(Vehicle):
    pass


class Van(Vehicle):
    pass


van1 = Van("141-DL-1234", "Ford", "Transit")
car1 = Car("201-D-4321", "Skoda", "Yeti")
```

Now I can do van1.get_county and get “Donegal” back as a return. This feature greatly reduces the amount of code needed and avoids repetition as functions can be reused among children.

Libraries

In today's smartphone world it's not uncommon to hear the phrase “there's an app for that”, well with Python you can say “there’s a library for that”. Python's standard library is robust and extensive and features the most used and useful packages. On top of the Python standard library there is a package manager called PIP. This allows for the easy installation of over 250,000 packages and contains contributions from over 470,000 people. This massive collection of packages puts Python as one of the main players of the artificial intelligence and machine learning field. Python has packages for everything from the “requests” library for HTTP functions, “numpy” for data scientists all the way to “rsa” to implement encryption methods.
The presence of all the modules make development in Python very quick. The developer can focus their time on solving the problems needed for their application not implementing “boiler plate” code.
These libraries are great for developers when coding but can cause issues when trying to deploy code. These libraries need to be installed on every system you intended to execute the code on. Libraries can receive updates causing problems if the same version is not used at all places you execute the code.

Summary & Conclusion

Python is truly a *"Jack of all trades, master of none!".* Python is one of, if not the easiest language to learn, this is a result of its indented (off-side) and simple syntax. It is one of the most versatile and useful languages available today.
But this versatility is also Python’s major downfall. For a large amount of use cases you can use Python, but should you? For web development languages such as Google's Go(Lang) exist which are optimized for concurrency and performance and can perform similar functions to Python but up to forty times the speed compared to Python. This is why companies such as YouTube are migrating away from Python towards languages such as Go. Python concurrency is primitive compared to Go as it wasnt written with concurrency in mind but as an afterthought. This is exacerbated by the fact Python is interrupted which adds a significant overhead compared to compiled languages. For GUI use cases languages such as Java and C# provide better tools for creating desktop applications.
Python strength comes from the fact it can accomplish all the above tasks, not with the same speed/ease but utilizing libraries all the above can be accomplished easily. Yes a developer could learn 5 different languages that are ‘better’ than Python at a given task or they could just learn Python and use it for all of them.
Speed truly is Python's biggest caviet, all other design choices made in the creation of Python such as the lack of static typing etc. are objective, they have pros and cons and can be better/worse depending on the use case. But Python's slow speed is significantly slower, to the point where it becomes increasingly costly to use for large programs dealing with thousands of clients and for time sensitive use cases. In the early days at Google a strategy of “Python where we can, C++ where we must” was introduced. This shows that speed hurdles can be overcome by outsourcing intensive tasks to lower level compiled languages when necessary and using Python for the rest due to its increased development speed.

Every language has its pros and cons, Python is no exception. But today at the end of 2020, Python is still a very good language to learn and will likely continue to improve via open source contributions to the Python source code and PIPs library catalogue.

References

[1] - https://docs.python.org/3/fIntroductionaq/general.html#what-is-python
[2] - Functional Programming: Proceedings of the 1989 Glasgow Workshop 21–23
[3] - https://www.computing.dcu.ie/%7Edavids/courses/CA341/CA341_assign.html#assignment_3
https://pypistats.org/top
[#] - https://docs.python.org

(Note: Hope these references are sufficient, first long essay where we had to reference, majority was just to get talking points no text transferred etc., but content is my own.)