01_IntroPython

July 10, 2019

Coursebook: Getting Started with Python - Part 1 of *Python Fundamental Course* - Course Length: 9 Hours - Last Updated: July 2019

- Developed by Algoritma's product division and instructors team

1 Background

The coursebook is part of the **Python Fundamental Course** prepared by Algoritma. The coursebook is intended for a restricted audience only, i.e. the individuals and organizations having received this coursebook directly from the training organization. It may not be reproduced, distributed, translated or adapted in any form outside these individuals and organizations without permission.

Algoritma is a data science education center based in Jakarta. We organize workshops and training programs to help working professionals and students gain mastery in various data science sub-fields: data visualization, machine learning, data modeling, statistical inference etc.

1.1 Training Objectives

On the first section of this **Python Fundamental Course**, we'll start by getting used in working with Jupyter Notebook and programming basics. In this coursebook we will cover:

- Object Oriented Programming
- Variable and Data Types
- Expression and Statements
- Style Guide

For the most part, experience in Python programming is good to have but not required. Familiarity with data manipulation and data structures in a different programming language a welcome addition but again, not required.

1.2 Object Oriented Programming

1.2.1 Class and Instance

Every object created in Python is generated from a class. You can imagine a class as a *blueprint* for creating an object. Let's refer to the `Inventory` class in `demo_oop.py` file:

```
In [5]: class Inventory:
            listed = False

            def __init__(self, name, price, discount):
                self.name = name
                self.price = int(price)
                self.discount = int(discount)
                self.headline = f"{name} at {self.net_price()} IDR today!"

            def net_price(self):
                return self.price - self.discount
```

This block code creates a *blueprint* named Inventory which we can use to create an **instance** of it and saved it in an object called `journal`:

```
In [11]: journal = Inventory("Tumbler", 50000, 3000)
```

The instance, is as might be able to guess is generated using the `__init__` function declared within the `Inventory` class. This way, everytime you need to create an instance, you can use `Inventory()` and pass in the required parameters.

1.2.2 Methods and Attributes

A class, as declared in the *blueprint* can also holds a certain attributes and methods. Let's try to understand each of the new term.

- Attribute is a variable property owned by the class. In our `Inventory` class, the attributes consist of `name`, `price`, `discount`, and `headline`.
- Method is a functional property owned by the class. In our `Inventory` class, we have `net_price()` as the method.

A method and attribute can only be accessed once we created an instance out of it:

```
In [12]: print(journal.net_price())

47000
```

```
In [13]: print(journal.headline)

Tumbler at 47000 IDR today!
```

Dive Deeper: We'll go ahead and refer to `demo_oop.py` file. Open it in the text editor of your choice!

1.3 Variable and Data Types

Now we have gone to unnderstanding how object oriented programming, let's take a step back and complete our fundamental building blocks: *variables*.

A variable is used in order for us to store a value. So the next time you need to use it, you will refer to the variable you have created before:

```
In [17]: age = 24

         print(2019 - age)
```

```
1995
```

1.3.1 Basic Data Types

As you work with Python, you will came accross several types of variable:

1. Boolean: `True` or `False`
2. Numeric: `int`, `float`, `complex`
3. Character: Alphanumeric and symbols

To return variable types, we can use `type()` function:

```
In [51]: print(type(age))
```

```
<class 'int'>
```

Dive Deeper:

Can you create a variable for each of the types and use `type()` to check if the variables you created results in the expected class?

Discussion:

Try to create a variable storing your name. Use `type()` on the variable, and it will result in `str`. What do you think an `str` stands for?

```
In [42]: # ====
         # YOUR ANSWER HERE
         # ===
```

A set of values stored in a variable can create a sequence. There are multiple basic data structure to choose form if we were to store a sequence value:

1. Tupple
2. List
3. Dictionary

```
In [2]: a_tuple = "Python", "R", "Julia"
        print(type(a_tuple))
        a_list = ["Python", "R", "Julia"]
        print(type(a_list))
```

```
<class 'tuple'>
<class 'list'>
```

The difference of tuple and list is tuple are not mutable or immutable while list is mutable. So what does this mean? This mean that you can change the value in a list, but you can't do that to a tuple. In a more advance implementation, you will see how it makes more sense to create a tuple over list and vice versa.

Lastly, as it name suggested, a Dictionary is what you will use if you need a, well, dictionary. It is used to represent a value and pair it with some unique ID:

```
In [49]: a_dict = {"name": ["Handoyo", "Fafilia"], "age":24}
         print(type(a_dict))
```

```
<class 'dict'>
```

```
In [50]: print(a_dict["name"])
```

```
['Handoyo', 'Fafilia']
```

Notice how we can use square bracket (`[]`) to take a *subset* of our dictionary. This method is called subsetting, and we will talk more in depth in the next section.

1.3.2 Slicing and Index

The technique of taking a *subset* from our sequence using element's index is called slicing. You might notice from previous example that you can use a string to specify an element. A more common use, however, is by using an element's index:

```
In [3]: a_list[0]
```

```
Out[3]: 'Python'
```

Notice how specifying `0` would result in the first element of our list. In Python, our indexing starts at the number `0`. This means, to access the third element we'll be using `2` as our subsetting index:

```
In [4]: a_list[2]
```

```
Out[4]: 'Julia'
```

Other related technique, is by using negative as our slicing index, we will get a backward indexing number:

```
In [56]: a_list[-1]
```

```
Out[56]: 'Julia'
```

Slicing list also works with iteration. By specifying a colon (`:`), we can specify the index which we want our slicing to start and end:

```
In [62]: a_list[0:2]
```

```
Out[62]: ['Python', 'R']
```

1.4 Expression and Statement

1.4.1 Basic Operators

Working with variables include a series of operation you can do with it. Start with a simple one: *arithmetic operation*. This operation is the most basic operation you can do with your variables. In the following chunk, try out basic arithmetic operations such as `+`, `-`, `/`, `//`, `%`, `**`:

```
In [6]: # Try out different operations here
```

Now moving on with the uncommon ones. In Python, we treat a *string* as a sequence of characters. We can also apply a string operations toward a string:

```
In [8]: brand = "Tokopedia"

        brand + " is the largest e-commerce in Indonesia"

Out[8]: 'Tokopedia is the largest e-commerce in Indonesia'
```

Remember how an object is generated through a class that holds methods and attributes? A string, same with any other object, also has a set of useful methods we can utilize. Let's explore some of them:

```
In [9]: brand.upper()

Out[9]: 'TOKOPEDIA'

In [12]: a_sentence = brand + " is the largest e-commerce in Indonesia"
         a_sentence.split()

Out[12]: ['Tokopedia', 'is', 'the', 'largest', 'e-commerce', 'in', 'Indonesia']

In [17]: a_sentence.replace("e-commerce", "startup")

Out[17]: 'Tokopedia is the largest startup in Indonesia'
```

Dive Deeper:
Try out different methods you can find within the string object! Remember to always work as closely as possible with Google and read up tons of documentation online!

1.4.2 Conditional Statement and Loops

Other fundamentals application in working with programming language is conditional statement. This application implements other types of operator: *logical* and *comparison*.

A logical operator is used to operate between booleans and consists of `and`, `or`, and `not`:

```
In [19]: print(True and True)
         print(True and False)

True
False
```

```
In [20]: print(True or False)
         print(False or False)
```

```
True
False
```

```
In [21]: print(not False)
```

```
True
```

While comparison operators is used to compare values between 2 objects:

```
In [23]: a = 2
         b = 3

         print(a == 2)
         print(a > 3)
         print(a != b)
```

```
True
False
True
```

This operators are applied to a conditional statements using `if`, a simple, yet powerful application of programming:

```
In [26]: jobs = ["Data Analyst", "Student", "Researcher", "Pilot"]

         for job in jobs:
             if job == "Data Analyst":
                 print(job + " might be the best job ever!")
             elif job == "Researcher":
                 print("Working as a "+ job + "? Cool!")
             else:
                 print("Learning how to work with data might be a good investment!")
```

```
Data Analyst might be the best job ever!
Learning how to work with data might be a good investment!
Working as a Researcher? Cool!
Learning how to work with data might be a good investment!
```

Pay attention on how the `for` can automatically iterate through the lists elements within `jobs`. This is a very handy tricks we are going to utilize a lot in the upcoming classes.

Discussion:

Have you heard of `while` loops? How does that differentiate from `for` loops? When do you use one instead of the other?

1.5 Functions

Defining functions can made a huge difference for our programming practice. It stands for 2 purpose: reproduciblity and modularity. Reproducibility means that a programming logic, can be saved as a function to reuse in the upcoming times. While modularity lean more in grouping a certain programming logic to a function to produce a better categorization for our codes.

To create a function we can use the `def` syntax. Recall the methods declared in `demo_oop.py` file:

```
In [29]: import datetime

         class User:
             def __init__(self, username, email):
                 self.username = username
                 self.email = email
                 self.timestamp = datetime.datetime.now()

             def signed_up_on(self):
                 delta = datetime.datetime.now() - self.timestamp
                 secs_since = int(delta.total_seconds())
                 return f"{self.username} signed up {secs_since} seconds ago."

         user_A = User("tiaradwptr", "tiara@algorit.ma.com")

In [30]: user_A.signed_up_on()

Out[30]: 'tiaradwptr signed up 15 seconds ago.'
```

A method, is basically a function stored within a Class. You can also create a stand-alone function object by using the same `def` syntax:

```
In [41]: def countDays(Day = datetime.datetime.now()):
             TimePass = Day - datetime.datetime(2019,1,1)
             return str(TimePass.days) + " days have passed since the beginning of 2019"

In [42]: countDays()

Out[42]: '190 days have passed since the beginning of 2019'
```

Dive Deeper:

Can you create a function to help you calculate number of hours and minutes have passed since you first join Tokopedia?

Congratulations, you have finished Getting Started with Python course! This course helps you to get a better idea of a fundamental building blocks in Python Programming. Our team at Algoritma had a nice time preparing the course material and hoped that this could help to kickstart your practice in working more effectively with Python.

For those of you that is new to programming, don't be discouraged with the amount of new knowledge you are learning within this course! As long as you are able to master the fundamental concepts, you will find an easier learning curve and find that everything you are doing in Python made complete sense!

Always find time to practice the things you learn today and don't be afraid to ask for help. Ask your mentors, friends, online forums, and our team of teaching assistant to help you get through the "hard part" in learning programming. Happy coding and good luck!