Python Language Basics

Recap Last Class

- Introduction into Object-Oriented Programming
  - Objects
    - State
    - Behaviour
  - Classes
- Python Language
  - Variables - Capture states
  - Methods - Capture behaviour

Questions from Last Class

- What is meant by an instance?
- Does a function have to have parameter passing?
- How to write decimals?
- Why are the “()” needed e.g. frank.move()?.
- Why do you need 2 different classes to be able to run the application? Why can’t you just do all the coding in 1 class?

Variables

- In the previous lesson, we saw the Class Agent had the variables:
  - `counter = 0`
  - `virtual = True`
- This code must have raised a few questions:
  1. What are the other data types in Python?
  2. What are the rules and conventions for naming variables?
  3. Do the variables need to be **initialized** (have an initial value) when declared?

Naming Convention for Variables

- You can use CamelCase, e.g. `thisIsMyVariableName`
- Variable names are **Case Sensitive**.
- White spaces is not allowed in variables.
- Try to use mostly letters.
- Avoid using “$”.
- Use full words instead of cryptic abbreviations.

Naming Convention for Variables

- This is a naming convention you see in many languages
- But it is not the default for Python
  - Naming convention for Python is this_is_my_variable, this_is_my_function
  - or ThisIsMyClass.
- Following PEP8 https://www.python.org/dev/peps/pep-0008/
- Different naming conventions are most common in different languages
  - It is most important to be consistent in your own program

Data Types

- All variables in Python are not declared with a type, e.g. **`int`** `Age`.
  This is not needed as all variables are objects and can change type Instantly.

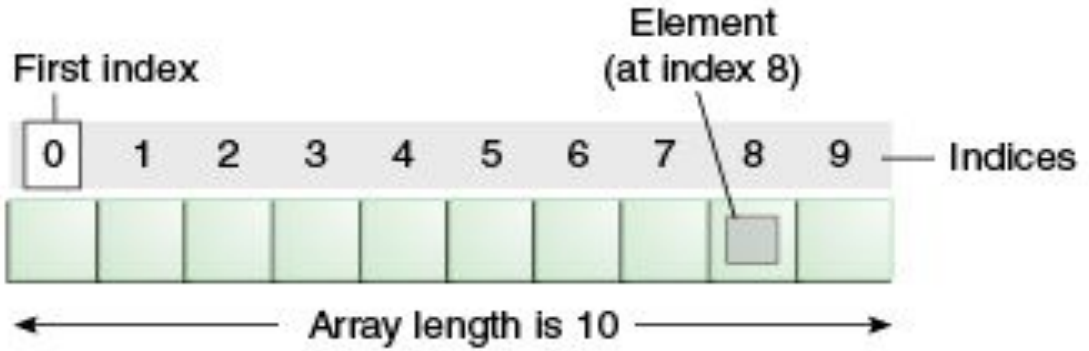
- Doing so will tell your program that a variable named "age" exists, but does not say what kind of data it holds.

```
Age = 19        Age = “nineteen”  Age = 19.0

>>> Age         >>> Age           >>> Age

19              “nineteen”        19.0
```

Data Types

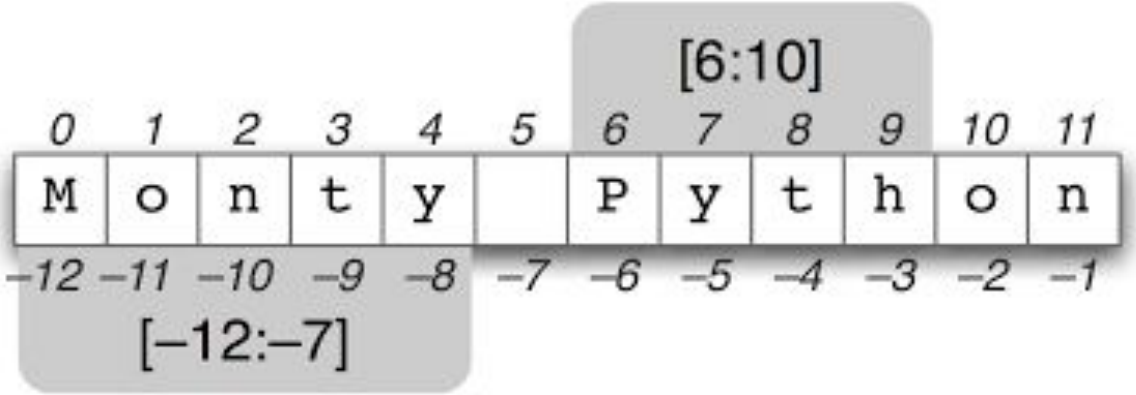
- You will use these data types in our lessons:
  - **`Int`** - Numerical value, `42325123`, etc.
  - **`Boolean`** - `True` or `False` values.
  - **`Float`** - `42.2345678901234567` decimal values.
  - **`String`** - `"This is a string"` value.
  - **`List`** - `[1,2,3,4]` value.
  - **`Dictionary`** - `{1:12,2:"hello",3:[1,2,3],4:25.10}` value.

Arrays, lists and dicts



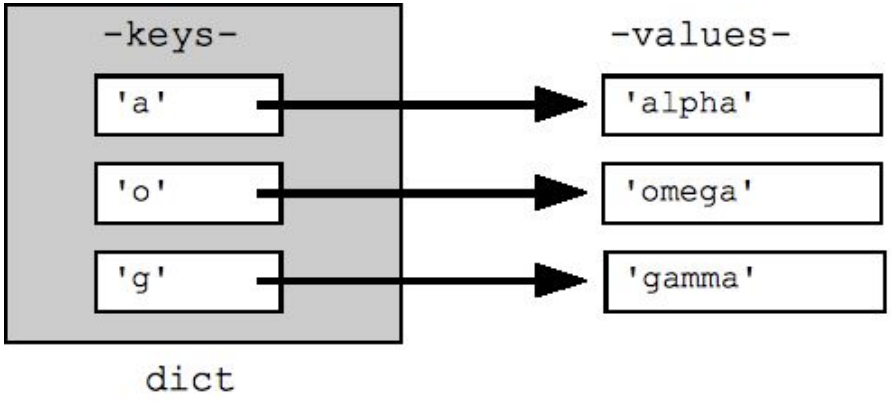
- An *array* is a container object that holds a number of values of a single type.
- The length of an array is established when the array is created.
- After creation, its length is fixed.

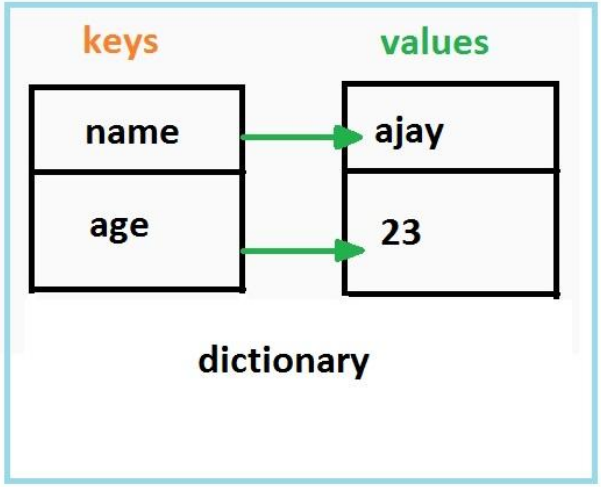
Arrays, lists and dicts



- Alist is a container object that holds a number of values of different types.
- The length of an list is **not** established when the list is created.
- After creation, its length is **not** fixed.

Arrays, lists and dicts





- An *dict* is a container object that holds a number of keys and values of different types.
- **Keys need to be the same type and cannot be lists!!**

List Code

```
class Agent_ListDemo(Agent):
   def __init__(self):
        self.path = []
   def move() :
         self.step(direction)
         self.path(direction)

  def print_path(self):
         for step_on_path in (path):
                  print(step_on_path)
```

Dict Code

```
class Agent_ListDemo(Agent):
   def __init__(self):
        self.intersection = {}
   def move() :
         if not self.location in self.intersection.keys():
                  self.intersection[self.location] = []
         self.intersection[self.location].append(direction_we_travelled_in)
```

Variables Kinds

- In the Python programming language there are FIVE kinds of variables:
  - Instance Variables
  - Class Variables
  - Local Variables
  - Global Variables
  - Parameters

These types are also referred to as the variable scope

- We will learn what these are in the following slides.

Instance Variable

- Instance Variables
  - This means their values are unique to each Instance of a class.
  - For example, in the Agent Class, **counter** is an instance variable.
  - Since the current counter of one agent is independent of and can be different from the counter of another agent.
- Another example of an Instance variable in our Agent class?

Class Variable

- Any variable declared outside a method but within a class is a class variable.
- This states that there is exactly one copy of this variable in existence for any number of instances created.
- For example, In the Agent class we can introduce a class variable to indicate the maximum counter for an Agent:
    - `maxCounter = 120`

Local Variable

- A Class may have many methods, and each method may store its own temporary states as local variables.
- There is no distinctive way of declaring a local variable. It is determined by its location within your code.
- Can you find a local variable within the Agent class code?
- Assignment: Change the Agent class to increase counter by 3. By making the Agent move for 3 steps

Global Variable

- Defined outside classes **AND** methods
- Can be changed by using the same variable name preceded by global within a function
- **USE AS LAST RESORT**

```
>>> def f():
...     print(x)
>>> def g():
...     global x
...     print(x)
...     x = 'g'
>>> x = 'global'
>>> f()
global
>>> g()
global
>>> f()
g
```

Parameter

- A parameter is a variable used to pass values into a method.
- You have already seen many examples of parameters.
- For example, in the Class Agent:
    - `def look(self, direction):`
- Assignment: Change the agent class - Move the agent for 3 steps in any direction that you prefer but it should not hit a wall.

Language Operators

Arithmetic Operators

- Python programming language provides operators that perform addition, subtraction, multiplication, and division

`+`	additive operator (also used for String concatenation)
`-`	subtraction operator
`*`	multiplication operator
`/`	division operator
`%`	remainder operator
`+=`	addition and assignment

ArithmeticDemo

```
class Agent_Arithmetics(Agent):

    def change_direction (self, direction):

         if direction == 3:
            direction = 0
         else:
            direction += 1
         return direction

     def move():
        direction = 0

        if self.look(direction) == -1:
           direction = self.change_direction(direction)
           self.step(direction)
```

Equality and Relational Operators

- The equality and relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand.

```
==      equal to
!=      not equal to
>       greater than
>=      greater than or equal to
<       less than
<=      less than or equal to
```

Equality and Relational Operators

- The following 3 are python specific

```
is      same as == (Python specific)

not     same as ! (Python specific)

in      to check whether a value exists in a list, dict or
        string
```

Unary Operators

- The unary operator requires only one operand.

```
+        Unary plus operator; indicates positive value

-        Unary minus operator; negates an expression

!, not   Logical complement operator; inverts the value
                    of a boolean

  Note that the increment/decrement operators(++ and --)
don’t exist in python
```

UnaryDemo

```
if __name__ == '__main__':
    unary_demo = UnaryDemo()
    unary_demo.demo()
```

```
class Agent_Unary(Agent):

    def demo(self):

        result = 1
        print(result)

        result += 1
        print(result)

        result = -result
        print(result)

        success = False
        print(success)
        print(!success)
```

Conditional Operators

- The && and || operators perform *Conditional-AND* and *Conditional-OR* operations on two boolean expressions.

```
&&, and Conditional-AND

||, or Conditional-OR
```

```
if sky is “blue” and time > 7 and time < 19:
    print(“it’s sunny”)
elif sky is not “blue” or not (time > 7 and time < 19):
    print(“it’s a wee bit dark”)
else:
    print(“I am hungry”)
```

ComparisonDemo

```
class Agent_if_else (Agent):
    def move(self):

        if self.look(1) == -1 and self.look(3) == -1:
            # there is no wall in front
            self.step(0)
        elif self.look(0) == -1 && self.look(3) == -1:
            # look right if there is no wall
            self.step(1)
        elif self.look(1) == -1 or self.look(0) == -1:
            # look left if there is no wall
            self.step(3)
        else:
            # we are trapped
            self.step(2) # move back
```

Break

Control Flow Statements

Control Flow Statements

- The statements inside your source files are generally executed from top to bottom, in the order that they appear.
- *Control flow statements*, however, break up the flow of execution

by employing decision making, looping, and branching

- This enables your program to *conditionally* execute particular blocks of code.

if-then Demo

```
if __name__ == '__main__':
    ifelse_demo = Agent_if_else()
    ifelse_demo.demo()
```

```
class Agent_if_else (Agent):
    def move(self):

        if not self.look(0) == -1:
            # there is no wall in front
            self.step(0)
        elif not self.look(1) == -1:
            # look right if there is no wall
            self.step(1)
        elif not self.look(3) == -1:
            # look left if there is no wall
            self.step(3)
        else:
            # we are trapped
            self.step(2) # move back
```

Assignment If-then-else

- Use the if-then-else statement in one of the methods of your agent Class

If-then-else or Switch

- Deciding whether to use if-then-else statements or a switch statement is based on readability and the expression that the statement is testing.
- An if-then-else statement can test expressions based on ranges of values or conditions, whereas a switch statement tests expressions based only on a **single** integer, enumerated value, or String object.

dict demo

```
class SwitchDemo(object):
    def get_month(self,month_i):

        switcher = {
        1: "January",
        2: "February",
        3: "March",
        4: "April",
        5: "May",
        6: "June",
        7: "July",
        8: "August",
        9: "September",
        10: "October",
        11: "November",
        12: "December"
    }
    print switcher.get(month_i, "Invalid
month")
```

This is a dictionary

```
if __name__ == '__main__':
    Switch_demo = SwitchDemo()
    Switch_demo.get_month(8)
    Switch_demo.get_month(13)
    Switch_demo.get_month(-1)
```

Looping

- Going round and round in circles.
- There are two different kinds of loops (for and while)
- While is done while a condition is true

```
x = 0
while (x < 10)
    x += 1
```

- For is run for as long as a condition is true

```
for (int x= 0; x < 10; x++)
```

Show a pycharm example in the debugger

Difference: while and do-while

- The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top.
- Therefore, the statements within the do block are always executed at least once, as shown in the following whileDemo program:

while Demo

```
class Agent_WhileDemo(Agent):
    def move(self):

        while (self.counter < 3) {
            # move forward for 3 steps
            self.step(0)
            counter += 1
```

```
class Agent_DoWhileDemo(Agent):
    def move(self):
        while True:
                self.step(0)
                if self.look(0) == -1:
                # if we hit a wall stop
                   break
```

Assignment while statement

- Use the while statement in one of the methods of your Agent Class

The for Statement

- The for statement provides a compact way to iterate over a range of values.
- Programmers often refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied.
- The general form of the for statement can be expressed as follows:

```
for i in range(start, end, increment):
    statement(s)
```

Python specific way

The for Statement

- Pay attention to stop criterium.
- Python for loops continue as long as index (i) is smaller than end criterium (< end)
- In some other languages it is possible to continue as long as index is smaller than or equal to end criterium (<= end)

for-statement Demo

```
class Agent_ForDemo(Agent):
    def move(self):
         for i in range(3):
              self.step(0)
```

Array or list and For loops

You can use for loops to cycle through the array index

```
For i in range(0, len(anArray)):
        print("Element at index " + str(i) + " is " + str(anArray[i]))
```

Or

```
For item in anArray:
        print("Element in array is " + str(item))
```

Or

```
For i, item in enumerate(anArray):
        print("Element at index " + str(i) + " is " + str(item))
```

Assignment For Loop statement

- Use the For Loop statements in one of the methods of your Agent Class