notebook 1. Python basics: Syntax

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object;
- Names start with a letter A--Z or a--z or an underscore (_) followed by zero or more letters, underscores and digits. i.e. Airplane, bear, _car, etc;
- Python does not allow punctuation characters such as @, $, and % within identifiers;
- Python is a case sensitive programming language. Thus, Basic and basic are two different identifiers in Python.

In [ ]:
```
this_is_cat = 'cat'
```

In [ ]:
```
ThisIsDog = 'dog'
```

In [ ]:
```
_this_is_underscore = '_'
```

In [ ]:
```
this_is_@n_error = 'error'
# because Python does not allow punctuation characters such as @, $, and % within identifiers;
```

In [ ]:
```
2nd_error_this_is = 'yoda style'
# Names start with a letter A--Z or a--z or an underscore (_)
```

Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

Method Description

- `and` A logical operator
- `as` To create an alias
- `assert` For debugging
- `break` To break out of a loop
- `class` To define a class
- `continue` To continue to the next iteration of a loop
- `def` To define a function
- `del` To delete an object
- `elif` Used in conditional statements, same as else if
- `else` Used in conditional statements
- `except` Used with exceptions, what to do when an exception occurs
- `False` Boolean value, result of comparison operations
- `finally` Used with exceptions, a block of code that will be executed no matter if there is an exception or not
- `for` To create a for loop
- `from` To import specific parts of a module
- `global` To declare a global variable
- `if` To make a conditional statement
- `import` To import a module
- `in` To check if a value is present in a list, tuple, etc.
- `is` To test if two variables are equal
- `lambda` To create an anonymous function
- `None` Represents a null value
- `nonlocal` To declare a non-local variable
- `not` A logical operator
- `or` A logical operator
- `pass` A null statement, a statement that will do nothing
- `raise` To raise an exception
- `return` To exit a function and return a value
- `True` Boolean value, result of comparison operations
- `try` To make a try...except statement
- `while` To create a while loop
- `with` Used to simplify exception handling
- `yield` To end a function, returns a generator

In [ ]:
```
True = 1
# error because True is a keyword
```

In [ ]:

```python
yield = 4
# error because yield is a keyword
```

In [ ]:

```python
try = 'try'
# error because try is a keyword
```

Comments and docstring

Comments are like signposts which make a given code self-evident and highly readable. In Python, we can add single-line (#, ‘’, “”) and multi-line Python comment (“”” “””, triple). Writing comments is a good programming practice. They are non-executable part of the code, yet quite essential in a program.

Documentation strings (or docstrings) provide a convenient way of associating documentation with Python modules, functions, classes, and methods. It’s specified in source code that is used, like a comment, to document a specific segment of code. Unlike conventional source code comments, the docstring should describe what the function does, not how. The docstrings are declared using “””triple double quotes””” just below the class, method or function declaration. All functions should have a docstring.

Call function docstring: `function_name.__doc__`

In [ ]:

```python
# this function is an example for comments and docstrings
def count_substring_in_string(substring, string):
    """counts number of substring in a string.

    Args:
        substring: a substring to search and count
        string: a text
    Returns:
        c: number of substring in a string
    """
    # counting length of substring
    n = len(substring)
    'initial count value is 0'
    c = 0
    "for loop to iterate over all string"
    for i in range(len(string)-n+1):
        """condition to compare substrings"""
        if substring == string[i:i+n]:
            # if True then increase counter
            c += 1
    # returning result
    return c
```

In [ ]:

```python
# calling function with arguments
count_substring_in_string("ABA", 'ABABABABA')
```

In [ ]:

```python
# calling docstring of the function
print(count_substring_in_string.__doc__)
```

Lines and Indentation

As you can see from the example function above Python provides no braces to indicate blocks of code for class and function definitions or flow control (as is in c++/java).

Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. (usually used 4, sometimes 2)

When block of code ends, the space indentation must go back to previous block value

In [ ]:

```python
 1  def count_substring_in_string_no_comments(substring, string):
 2      n = len(substring)
 3      c = 0
 4      for i in range(len(string)-n+1):
 5          if substring == string[i:i+n]:
 6              c += 1
 7      return c
 8
 9
10  # Line 1: function declaration
11  # Lines 2-7: function block
12  # Lines 5-6: subblock of `for` loop
13  # Line 6: subblock of `if` conditional
14  # Line 7: return result of the function, as we can see the intendation of 7th row goes back to level of lines 2-4,
15  #         that indicates that this line lies outside of previous block
16  #         and comes to function block
```