College Logo

# Lecture notes on Python Programming

**By Jason Pandian**

*Assistant Professor, Department of Information Technology*

# DATA TYPES, CONTROL FLOW, STRINGS

Control Flow -conditional (if), Alternative (if-else), Chained conditional (if-elif-else)- Iteration: state, while, for, break, continue, pass - Strings: string slices, immutability, string functions and methods, string module, Regular expression, Pattern matching . - Illupdated_strative Problems.

## What is Control Flow?

Control flow in Python refers to the order in which statements and instructions are executed in a program. It determines the flow of execution based on certain conditions or loops.

## If Statement

Used for executing a block of code if a specified condition is True.

In [1]:
```python
x = 10
if x > 0:
    print("x is positive")
```

```
x is positive
```

## If-Else Statement

Executes one block of code if the condition is True, and another block if it's False.

In [2]:
```python
y = -5
if y > 0:
    print("y is positive")
else:
    print("y is non-positive")
```

```
y is non-positive
```

## Chained Conditional (If-Elif-Else)

Allows checking multiple conditions in sequence.

```
In [3]:
z = 0
if z > 0:
    print("z is positive")
elif z < 0:
    print("z is negative")
else:
    print("z is zero")
```

```
z is zero
```

## Iteration

Iteration in programming is the repetition of a set of instructions or code block until a certain condition is met.

### State

- "state" - **status** refers to the current values of variables and attributes within an object or program, representing its configuration at a given moment.
- It encapsulates data such as variable values, object attributes, and program execution status, crucial for understanding and managing program behavior and interactions.

### While Loop

Repeats a block of code as long as a given condition is True.

```
In [4]:
count = 0
while count < 5:
    print("Count:", count)
    count += 1
```

```
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
```

```
In [5]:
count = 0
while count <= 5:
    print("Count:", count)
    count += 1
```

```
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

In [6]:
```
count = 5
while count >= 0:
    print("Count:", count)
    count -= 1
```

```
Count: 5
Count: 4
Count: 3
Count: 2
Count: 1
Count: 0
```

In [ ]:
```
while 1 :
    print("True")
```

In [ ]:
```
count = 5
while count >= 0:
    print("Count:", count)
    count += 1 # error count = count + 1
```

## For Loop

Iterates over a sequence (e.g., list, tuple, string) or other iterable objects.

In [1]:
```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

```
apple
banana
cherry
```

In [3]:
```
for i in range(1,11):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

## Break Statement

Terminates the loop prematurely when a certain condition is met.

In [8]:
```
for num in range(10):
    if num == 5:
        break
    print(num)
```

```
0
1
2
3
4
```

## Continue Statement

Skips the rest of the code inside the loop for the current iteration and moves to the next one.

In [9]:
```
for num in range(5):
    if num == 2:
        continue
    print(num)
```

```
0
1
3
4
```

## Pass Statement

Acts as a placeholder; it does nothing and is used when a statement is syntactically required.

In [11]:
```
for item in fruits:
    # Some code here
    pass  # Placeholder, does nothing
```

## Strings

In Python, strings are **sequences of characters, immutable**, and can be manipulated using various built-in **functions and methods**.

## String Traversal

In [8]:
```
for c in "Hello, World":
    print(c)
```

```
H
e
l
l
o
,
 
W
o
r
l
d
```

## String Slicing

```
In [78]: my_str = "Python Lang"
         print(my_str[0])    # Output: 'P'
```

```
P
```

```
In [77]: print(my_str[0:1])    # Output: 'P'
```

```
P
```

```
In [64]: print(my_str[0:5])    # Output: 'Pytho'
```

```
Pytho
```

```
In [65]: print(my_str[0:6])    # Output: 'Python'
```

```
Python
```

```
In [66]: print(my_str[6:7])    # Output: ' '
```

```
 
```

```
In [67]: print(my_str[7:8])    # Output: 'L'
```

```
L
```

```
In [68]: print(my_str[-1:])    # Output: 'g'
```

```
g
```

```
In [69]: print(my_str[:])      # Output: 'Python Lang' (Full string)
```

```
Python Lang
```

```
In [70]: print(my_str[:6])     # Output: 'Python' (From the beginning up to index 6)
```

```
Python
```

```
In [71]: print(my_str[7:])     # Output: 'Lang' (From index 7 to the end)
```

```
Lang
```

```
In [72]: print(my_str[-4:])    # Output: 'Lang' (Last 4 characters)
```

```
Lang
```

```
In [83]: print(my_str[::2])    # Output: 'Pto ag' (Every other character) ->stepsize
```

```
Pto ag
```

```
In [74]: print(my_str[::-1])   # Output: 'gnaL nohtyP' (Reverse the string)
```

```
gnaL nohtyP
```

```
In [4]: # Example of strings as sequences of characters
        my_string = "Hello, World!"
        print("Characters in the string:", my_string)  # Output: Hello, World!

        # Accessing individual characters in the string
        print("First character:", my_string[0])  # Output: H
        print("Last character:", my_string[-1])  # Output: !
```

```
Characters in the string: Hello, World!
First character: H
Last character: !
```

```
In [3]: # Example of string immutability
        # Attempting to modify a character in the string will result in an error
        my_string[0] = 'h'  # Raises TypeError: 'str' object does not support item a
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 3
      1 # Example of string immutability
      2 # Attempting to modify a character in the string will result in an e
rror
----> 3 my_string[0] = 'h'  # Raises TypeError: 'str' object does not suppor
t item assignment

TypeError: 'str' object does not support item assignment
```

```
In [6]: # String manipulation example
        # Replacing a substring
        modified_string = my_string.replace("Hello", "Hi")
        print("Modified string:", modified_string)  # Output: Hi, World!
        type(modified_string)
```

```
Modified string: Hi, World!
```

```
Out[6]: str
```

```
In [3]: # Splitting the string into a list of substrings
        split_string = my_string.split(',')
        print("Split string:", split_string)  # Output: ['Hello', ' World!']
        type(split_string)
```

```
Split string: ['Hello', ' World!']
```

```
Out[3]: list
```

```
In [7]: # Joining the substrings back into a single string
        joined_string = ','.join(split_string)
        print("Joined string:", joined_string)  # Output: Hello, World!
        type(joined_string)
```

```
Joined string: Hello, World!
```

```
Out[7]: str
```

## Key Takeaway

- Sequence of characters.
- Immutable.
- But can be manupulated

## Under the hood for Simple Replace

```
In [5]: my_string = "Hello, World!"
        print(len(my_string))
```

```
13
```

```
In [6]: def strReplace(my_string, given_str, updated_str):
            # Initialize an empty string to store the modified string
            new_string = ""

            # Initialize an index variable to iterate through the original string
            i = 0

            print("length of th given_str: ",len(given_str))

            # Iterate through the original string
            while i < len(my_string):
                # Check if the current substring matches the substring to be replace
                if my_string[i:i + len(given_str)] == given_str:
                    # If yes, append the replacement substring to the new string
                    new_string += updated_str
                    # Update the index to skip the replaced substring
                    i += len(given_str)
                else:
                    # If no match, append the current character to the new string
                    new_string += my_string[i]
                    # Move to the next character in the original string
                    i += 1

            # Return the modified string
            return new_string

        # Test the function
        print(strReplace("Hello, World!", "Hello", "hi"))
```

```
length of th given_str:  5
hi, World!
```

```
In [ ]: "hello" + " " + "world"
```

## String functions and methods

In Python, string functions and methods are tools used to manipulate strings, which are sequences of characters.

## String Functions

Definition: String functions are standalone functions that operate on strings and return a result. They do not modify the original string. Example: len() is a string function that returns the length of a string.

### len()

- `len()` : Returns the length of a string.

In [15]:
```
string = "Hello"
print(len(string))  # Function Example: 5
```

```
5
```

## String Methods

Definition: String methods are functions that are called on string objects and operate directly on them. They can modify the original string and/or return a modified version of it. Example: .lower() is a string method that converts all characters in a string to lowercase.

### 1. lower()

`lower()` : Converts all characters in a string to lowercase.

In [16]:
```
string = "hello"
print(string.upper())  # Method Example: HELLO
```

```
HELLO
```

### 2. upper()

`upper()` : Converts all characters in a string to uppercase.

In [17]:
```
string = "HELLO"
print(string.lower())  # Method Example: hello
```

```
hello
```

### 3. strip()

`strip()` : Removes leading and trailing whitespace from a string.

In [18]:
```
string = "   hello   "
print(string.strip())  # Method Example: hello
```

```
hello
```

## 4. split()

`split()` : Splits a string into a list of substrings based on a delimiter.

```
In [19]: string = "apple,banana,orange"
         fruits = string.split(',')
         print(fruits)  # Method Example: ['apple', 'banana', 'orange']
```

```
['apple', 'banana', 'orange']
```

## 5. join()

`join()` : Joins elements of an iterable into a string, using the string as a separator.

```
In [20]: fruits = ['apple', 'banana', 'orange']
         string = ','.join(fruits)
         print(string)  # Method Example: apple,banana,orange
```

```
apple,banana,orange
```

## 6. replace()

`replace()` : Replaces occurrences of a substring with another substring.

```
In [21]: string = "Hello, World!"
         new_string = string.replace("World", "Universe")
         print(new_string)  # Method Example: Hello, Universe!
```

```
Hello, Universe!
```

## 7. startswith()

`startswith()` : Checks if a string starts with a specified prefix.

```
In [22]: string = "Hello, World!"
         print(string.startswith("Hello"))  # Method Example: True
```

```
True
```

## 8. endswith()

`endswith()` : Checks if a string ends with a specified suffix.

```
In [23]: string = "Hello, World!"
         print(string.endswith("World!"))  # Method Example: True
```

```
True
```

### 9. find()

`find()` : Searches for a substring within a string and returns the lowest index where it's found.

In [24]:
```
string = "Hello, World!"
print(string.find("World"))  # Method Example: 7
```

```
7
```

### 10.count()

`count()` : Counts the number of occurrences of a substring within a string.

In [25]:
```
string = "apple, banana, apple, orange"
print(string.count("apple"))  # Method Example: 2
```

```
2
```

### 11. capitalize()

`capitalize()` : Converts the first character of a string to uppercase and the rest to lowercase.

In [26]:
```
string = "hello world"
print(string.capitalize())  # Method Example: Hello world
```

```
Hello world
```

### 12. casefold()

`casefold()` : Converts a string to lowercase, suitable for case-insensitive comparisons.

In [27]:
```
string = "Hello World"
print(string.casefold())  # Method Example: hello world
```

```
hello world
```

### 13. isdigit()

`isdigit()` : Checks if all characters in a string are digits.

In [28]:
```
string = "123"
print(string.isdigit())  # Method Example: True
```

```
True
```

### 14. islower()

`islower()` : Checks if all characters in a string are lowercase.

In [29]:
```
string = "hello"
print(string.islower())  # Method Example: True
```
```
True
```

## 15. isupper()

`isupper()` : Checks if all characters in a string are uppercase.

In [30]:
```
string = "HELLO"
print(string.isupper())  # Method Example: True
```
```
True
```

## 16. isspace()

`isspace()` : Checks if all characters in a string are whitespace.

In [31]:
```
string = "    "
print(string.isspace())  # Method Example: True
```
```
True
```

## 17. istitle()

istitle(): Checks if a string is titlecased (i.e., every word starts with an uppercase character and the rest are lowercase).

In [32]:
```
string = "Hello World"
print(string.istitle())  # Method Example: True
```
```
True
```

## 18. split()

`split()` : Splits a string into a list of substrings based on a delimiter (default is whitespace).

In [33]:
```
string = "apple,banana,orange"
fruits = string.split(',')
print(fruits)  # Output: ['apple', 'banana', 'orange']

string = "apple,banana,orange"
fruits = string.split('a')
print(fruits)  # Output: ['apple', 'banana', 'orange']
```
```
['apple', 'banana', 'orange']
['', 'pple,b', 'n', 'n', ',or', 'nge']
```

## 19.isalnum()

`isalnum()` : Checks if all characters in a string are alphanumeric (either alphabetic or numeric).

```
In [34]: string = "Hello123"
         print(string.isalnum())  # Output: True
```

```
True
```

### 20. isascii

`isascii()` : Checks if all characters in a string are ASCII characters.

```
In [35]: string = "Hello"
         print(string.isascii())  # Output: True
```

```
True
```

```
In [36]: string = "Привет"
         print(string.isascii())  # Output: False
```

```
False
```

## String Module

The `string` module in Python provides a collection of useful constants and classes for working with strings. It is part of the Python Standard Library, which means it is available in all Python installations without the need for additional installation steps.

- Constants: The module includes several constants such as ascii_lowercase, ascii_uppercase, ascii_letters, digits, and punctuation. These constants provide predefined sets of characters that are commonly used in string manipulation tasks.
- Functions: The string module does not include standalone functions. However, you can use the constants provided in combination with other Python functions to perform various string operations.

```
In [37]: import string

         # Accessing predefined sets of characters
         print(string.ascii_lowercase)  # Output: abcdefghijklmnopqrstuvwxyz
         print(string.digits)           # Output: 0123456789
         print(string.punctuation)      # Output: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

```
abcdefghijklmnopqrstuvwxyz
0123456789
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

### Examples

#### Generating Random Strings

You can use the string.ascii_letters, string.digits, and other constants to generate random strings for tasks like generating passwords or tokens.

```
In [38]: import string
         import random

         # Generate a random password of length 8
         password = ''.join(random.choices(string.ascii_letters + string.digits, k=8)
         print(password)
```

```
UYmKAqQX
```

### Validating User Input

You can use constants like string.digits to check if a user input contains only numeric characters.

```
In [39]: import string

         def is_numeric(input_string):
             for char in input_string:
                 if char not in string.digits:
                     return False
             return True

         print(is_numeric("123"))  # Output: True
         print(is_numeric("123a")) # Output: False
```

```
True
False
```

```
In [40]: import string

         def is_numeric(input_string):
             return all(char in string.digits for char in input_string)

         print(is_numeric("123"))  # Output: True
         print(is_numeric("123a")) # Output: False
```

```
True
False
```

**Any Questions?**

## For More

**Refer the Lectures/Tutorials GitHub Page**