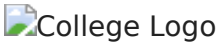

Lecture notes on Python Programming

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

If-Else Statement

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

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

Chained Conditional (If-Elif-Else)

Allows checking multiple conditions in sequence.

In [ ]:
```
z = 0
if z > 0:
    print("z is positive")
elif z < 0:
    print("z is negative")
else:
    print("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 [ ]:
```
count = 0
while count < 5:
    print("Count:", count)
    count += 1
```

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

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

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

In [ ]:
```
count = 5
while count >= 0:

```

```
    print("Count:", count)
    count += 1 # error
```

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]:
```python
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]:
```python
for c in "Hello, World":
    print(c)
```

```
H
e
l
l
o
,

W
o
r
l
d
```

String Slicing

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

```
P
```

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

```
P
```

In [64]:
```python
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 [2]:
```
# 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 [5]:
```
# 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[5]:
```
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 [46]:
```
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 [44]:
```
"hello" + " " + "world"
```

Out[44]: 'hello world'

Any Questions?

For More

Refer the Lectures/Tutorials GitHub Page