BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Political Analysis with Python

Öner Yigit

Week 2

July 16, 2023

Binghamton University (SUNY)

Agenda

- Control flow (if, elif, else)
- Loop (for, while)
- List comprehension
- Functions
- Lambda
- Methods
- Common methods

Section 1

Control flow

We often want a certain code to run when a specific condition is met. To control the flow of our code we have three keywords: if, elif and else.

`if` is used to evaluate an expression if a condition is `True`. It is the primary conditional statement. If the condition is `false`, the code block is skipped.

`elif` stands for **"else if"** and allows you to check additional conditions if the preceding if statement or elif statements are `False`.

`else` is used to specify the code block that should be executed if all preceding conditions (**if** and **elif**) are `False`. An `else` block does not have a condition and serves as a catch-all option.

Example 1

```
if condition1:
    # code block to execute if condition1 is true
elif condition2:
    # code block to execute if condition1 is false
    ↪   and condition2 is true
elif condition3:
    # code block to execute if condition1 and
    ↪   condition2 are false and condition3 is true
...
...
else:
    # code block to execute if all conditions are
    ↪   false
```

Example 2

```python
x = 10
if x < 5:
    print("x is less than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is greater than 5")
```

Example 3

```python
number = 8
if number % 2 == 0:
    print("Number is even.")
else:
    print("Number is odd.")
```

Indentation

Colons(:) and indentation(whitespace) are important. The expression should be indented either using four whitespace or one tab.

Once indentation is removed, conditions and executions cannot be identified.

Example 4

```
number = 36
if number % 2 == 0:
    print("Number is even.")
    if number % 3 == 0:
        print("Number is divisible by 6.")
    else:
        print("Number is not divisible by 6.")
else:
    print("Number is odd.")

'Number is even.'
'Number is divisible by 6.'
```

`While` Loop

`While` loops will continue to execute a block of code while some condition remains `True`.

For example, while my car is not full, keep filling my tank with gas.

While you are still a student, keep taking courses.

Example

```python
count = 1
while count <= 10:
    print(count)
    count += 1
```

This code will print all counts. So once `count=11`, the loop will terminate because `while` condition is no longer `True`.

Example

```
# smallest number greater than 700 divisible by 13
number = 700
while not number % 13 == 0:
    print(number, "is not divisible by 13.")
    number = number + 1
print(number, "is divisible by 13.")

'700 is not divisible by 13.'
'701 is not divisible by 13.'
'702 is divisible by 13.'
```

`For` Loop

Most Python objects are “iterable”, meaning we can iterate over every element in the object.

`for` loop and `while` loop can do the same thing. But `for` is much easier when number of iterations are known.

We can use for loops to execute a block of code for every iteration.

Example

```python
numbers = [1, 2, 3, 4, 5]

for number in numbers:
    print(number)
```

Example

```python
text = "Hello, World!"
vowel_count = 0

for char in text:
    if char.lower() in "aeiou":
        vowel_count += 1

print("Number of vowels:", vowel_count)
```

List Comprehensions

A unique way of quickly creating a list with Python.

It's possible to create a list, set, or dictionary using `for`, `while`, `if` and `else` in loops.

If you find yourself using a for loop along with `.append()` to create a list, List Comprehensions are a good alternative!

List comprehensions provide simple syntax to achieve it in a single line.

It is easy to read.

Example

```python
# Example 1: Squaring numbers using list
↪   comprehension
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x ** 2 for x in numbers]
print(squared_numbers)
```

Example

```
#simple example: create a copy of a list
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

#let's do with a for loop:
new_list = []
for number in numbers:
    new_list.append(number)
print(new_list)
```

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
#do the same with list comprehension
new_list = [num for num in numbers]
print(new_list)
```

Example

```python
# Filtering even numbers using list comprehension
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = [x for x in numbers if x % 2 == 0]
print(even_numbers)
```

Example

```
countries = ['USA', 'Canada', 'Mexico']
capitals = ['Washington, DC.', 'Ottawa', 'Mexico
↪   City']
capital_dict = {}
for i in range(len(countries)):
    capital_dict[countries[i]] = capitals[i]
```

```
capital_dict = {}
for country, capital in zip(countries, capitals):
    capital_dict[country] = capital
print(capital_dict)
```

Section 2

Functions

Creating clean repeatable code is a key in mastering Python.

Functions allow us to create blocks of code that can be easily executed many times, without needing to constantly rewrite the entire block of code.

if you repeatedly copy and paste your code, there's a problem

reuseable pieces of code

functions are not run until they are called / invoked somewhere

Function characteristics

- name
- parameters/arguments (*args and **kwargs)
- docstring (optional but recommended)
- body (lines of code that the function execute)
- returns something

`def`

Creating a function requires a very specific syntax, including the def keyword, correct indentation, and proper structure.

Let's get an overview of a Python function structure.

Example

```python
def is_even(i):
    """
    Input: i, a positive integer
    Returns True if i is even, otherwise False
    """
    return i % 2 == 0

is_even(5)
```

Example

```python
def calculate_rectangle_area(length, width):
    """
    Calculates the area of a rectangle.
    Parameters:
    - length: The length of the rectangle.
    - width: The width of the rectangle.
    Returns:
    - The area of the rectangle.
    """
    area = length * width
    return area
calculate_rectangle_area(5, 10)
```

Combine all

It is very important to get practice combining everything you've learned so far (control flow, loops, etc.) with functions to become an effective coder.

Learning functions increases your Python skills exponentially.

Example

```python
def check_number(number):
    if number > 0:
        return "positive"
    elif number < 0:
        return "negative"
    else:
        return "zero"

check_number(-3)
```

Example

```python
def calculate_rectangle_properties(length, width):
    if length <= 0 or width <= 0:
        return None, None, None

    perimeter = 2 * (length + width)
    area = length * width
    diagonal = (length ** 2 + width ** 2) ** 0.5

    return perimeter, area, diagonal
```

`lambda`

`lambda` function is an anonymous function

It is a way to define small, one-line functions without a formal function declaration.

Lambda functions are often used when you need a simple function that will be used once and does not require a separate definition.

Example

```python
addition = lambda x, y: x + y
result = addition(3, 5)
print(result)


(lambda x: x * 10 if x > 10 else (x * 5 if x < 5 else x))(11)


lst = [33, 3, 22, 2, 11, 1]
filter(lambda x: x > 10, lst)
```

Example

```
import pandas as pd
df = pd.DataFrame({'col1': [1, 2, 3, 4, 5], 'col2':
    [0, 0, 0, 0, 0]})
print(df)
df['col3'] = df['col1'].map(lambda x: x * 10)
df

#or we can use apply() function

df['col3'] = df['col1'].apply(lambda x: x * 10)
df
```

Function or Methods

- The terms “methods” and “functions” are related but have some distinctions.
- Functions are standalone blocks of code that can be called from anywhere
- Methods are functions associated with specific objects or classes.
- Methods are called on instances or objects of the class, and they have access to the object’s data and other methods.

Ten built-in function

- `print()`: Used to display output
- `input()`: Reads input from the user
- `len()`: Returns the length or number of items.
- `range()`: Generates a sequence of numbers.
- `type()`: Returns the type of an object.
- `str()`, `int()`, `float()`: Converts objects to one another.
- `list()`, `tuple()`, `dict()`: Converts objects to one another.
- `open()`, `read()`, `write()`: Methods for file handling.
- `sorted()`: Returns a sorted version.
- `sum()`: Returns the sum of all elements in a collection.
- `max()`, `min()`: Returns the maximum or minimum value.
- `abs()`: Returns the absolute value.
- `round()`: Rounds a number to a specified decimal place.
- `zip()`: Combines multiple iterables.
- `enumerate()`: Returns an iterator, the index and element.

`print()`

```
print("Hello, World!")
```

`input()`

```python
name = input("Enter your name: ")
print("Hello, " + name)
```

`len()`

```python
numbers = [1, 2, 3, 4, 5]
length = len(numbers)
print("Length:", length)
```

`range()`

```python
for num in range(1, 6):
    print(num)
```

Data type conversion

```python
x = 5
print(type(x))

num_str = str(5)
print(num_str)

my_tuple = (1, 2, 3)
my_list = list(my_tuple)
print(my_list)
```

append()

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)
```

`split()`

```python
sentence = "Hello, world! Welcome to Python."
words = sentence.split(" ")
print(words)
```

join()

```
words = ['Hello', 'world!', 'Welcome', 'to',
↪   'Python.']
sentence = ' '.join(words)
print(sentence)
```

`sorted()`

```python
my_list = [3, 1, 4, 1, 5, 9, 2, 6, 5]
sorted_list = sorted(my_list)
print(sorted_list)
```

enumerate()

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

for index, fruit in enumerate(fruits):
    print(index, fruit)
```

Next week

Of course, you cannot memorize all these, and you should not.

Practice will be the key. Play around with built-in functions.

Next week, we will look at data analysis modules/packages. Some methods will also be part of these packages.