Control Flow

Introduction to Data Science with Python

Outline

- Control Flow
  - if, elif, else
  - for, while
  - break, continue
- Exception Handling
- Comprehensions

Control Flow

- Truth testing
  - If something is true (or false), do this task. If the opposite is true, do something else…
  - if, elif, else
- Iteration
  - Iterate through a sequence, and do something at each iteration

    for loop
  - Continue to execute this code until…

    while loop

White Space Matters

- In python, white space is significant and implies the various control flow structures.
- Use 4 spaces to indent your code

  You can set the IDE to use 4 spaces for the tab function

```
for i in range(10):
    print(i, i+1)
    if i < 5:
        print("We have a small i")

# This code has a syntax error for indenting
for i in range(10):
    print(i, i+1)
  if i < 5:
 print(i, i+1)
```

If, Else, and Both

- Truth Testing

  If something is true do this; otherwise do that

```
# If Else elif
if 2 < 3:
    print(“The world is as it should be." )
else:
    print(“Something is wrong.")
>> The world is as it should be.


# On one Line
print(“Sounds correct.") if 2 < 3 else “I don’t think so."
```

Multiple Nesting

- We need to use pass sometimes as a placeholder

```
# Nesting
x = -5
if x > 0:
    print("Positive")
elif x == 0:
    print(“Zero")
elif x < 0:
    print("Negative")

>> Negative
```

Test Your Knowledge – Ex 5.1

- Now it's your turn to apply what you have learned.

Write Python code to do the following:

1. Prompt for input "Enter a number: "
2. If the user entered a negative number print "Please enter a positive number next time"
3. Otherwise output "The square root of the number is: " and provide the square root of the number.

for Loop

- For *item* in *collection*; do some stuff

  The traditional for loop. Can iterate through an object

```
# For loops
for i in "long string":
    if i != "n":
        print(i)
```

Test Your Knowledge – Ex 5.2

- Now it's your turn to apply what you have learned.

```
Write Python code to do the following:

1. Create a for loop that sums the numbers 1 through 10

2. Print the total
```

Comparison Operators

- Traditional comparison operators are supported:

  < , <=, > , >=, == , !=

  Note: Numbers are "smaller" than strings

  Case matters

```
# Comparison Operators
3 < 4
4 < 3
"the" < "thz"
4 < "the"
"the" == "THE"
"the" == "THE".lower()
"the" != "something else"
```

while

- Use a while loop to repeat a code-block as long as the condition is true

  while *condition*:

    statements…

```
# while loop example
x = 7
while x > 0:
    print(x)
    x = x - 1  # note we should modify condition somewhere

>> 7 6 5 4 3 2 1
```

break and continue

- Python provides two commands to alter the execution of a while loop
  - **break** – drop out of the loop to the next command below the loop
  - **continue** – Go back to the first command of the loop skipping any commands between the continue statement and end of the loop

```
x = 7
while x > 0:
    if x == 1:
        break
    if x > 5:
        continue
    print(x)

>> 5 4 3 2
```

pass

- Sometimes you need a no-op if only to serve as a placeholder
- The pass statement indicates "take no action"
  Often applied to maintain the proper use of whitespace

```
# Nesting and pass
x = -5
if x > 0:
    print("Positive“)
elif x == 0:
    #forgot
    pass
elif x < 0:
    print("Negative“)

>> Negative
```

Test Your Knowledge – Ex 5.3

- Now it's your turn to apply what you have learned. (Hint – take a look at the modulo operator %)

```
Write Python code to do the following:

1. Create while loop that adds the numbers 1 through 100

2. Put in a condition in the while loop that if the sum at any
   point is divisible (completely) by 5 then exit the loop

3. Print the resulting sum
```

Exceptions (Errors)

- Lets consider two types of errors:
- Syntax Errors
  - These arise because of incorrect coding syntax
  - We fix these with correct syntax ☺
- Errors may arise in function calls, user inputted data, or loops
  - These exceptions can be "caught", and we can ask Python to respond to these exceptions in certain ways.
  - If we don't, an exception will cause a "fatal" error

Exception Handling

- Another way to modify control flow by catching *specific* exceptions
  - Use the try…except construct

```
# Syntax Error
int(1 2)
>> SyntaxError: invalid syntax

# Specific Exceptions
int("1 2")
>> ValueError: invalid literal for int() with base 10: '1 2'

try:
    int("1 2")
except ValueError:
    print("1 2“)
```

Exception Handling

- By removing the exception specification we can generalize the exception to all exceptions and "catch" any kind of error

  How might you ignore an exception (warning: recent slide applicable…)? Can you think of a case?

```
# Non-specific Exception
a = 10
b = “a string"
try:
    print(a + b)
except Exception as e:
    print(“There was an error...\n", e)
```

Test Your Knowledge – Ex 5.4

- Now it's your turn to apply what you have learned.

Redo your square root Python code to do the following:

1. Prompt for input "Enter a number: "
2. Use an exception handler to ensure you can take the square root of the input
3. If there is an exception print "Unable to calculate the square root."
4. Otherwise output "The square root of the number is: " and provide the square root of the number.

Extra Credit – Can you put this in a loop so the user can reenter a number that is bad? How would you handle a string entry?

Comprehensions

- A “beloved" feature of python, but confusing too

  Suppose you wanted to create a list from an existing list by filtering it with a function

  Here’s one way to do it, but it isn’t very efficient

```
# List Comprehension Pre-cursor
list1 = range(10)
list2 = []
for i in list1:
    list2.append(i + 1)
list2
```

List Comprehension

- Python provides a shortcut for this common operation

  [*expr* for *val* in *collection* if *condition*]

```
# List Comprehension
list2 = [i+1 for i in list1]
>> [1, 2 ,3 ,4 ,5 , 6, 7, 8, 9, 10]

list3 = [i+1 for i in list1 if i > 4]
>> [6, 7, 8, 9, 10]

list4 = [i+1 if i > 4 else i for i in list1]
>> [0, 1, 2, 3, 4, 6, 7, 8, 9, 10]
```

Test Your Knowledge – Ex 5.5

- Now it's your turn to apply what you have learned.

1. Create a list of five elements from the numbers 1 through 5

2. Use comprehension to multiple each element by 3