04.1_Conditionals_Strings

September 11, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang

Content:

- Conditional Execution
- Strings

2 Conditionals with `if` statements

- Give us the ability to check conditions and change the behavior of the program accordingly.
- Check `True` or `False`
- Intense use of logical operations or comparison operations
- One of the five components of a program: input, output, conditions, repetition, math

2.0.1 (1) If statement on its own:

[1]:
```
x = 0
```

[2]:
```
x
```

[2]: 0

[3]:
```
x == 0
```

[3]:
```
True
```

[4]:
```
if x > 0:
    print('x is positive')
```

[5]:
```
x = -1
if x > 0:
    print('x is positive')
```

[6]:
```
x = 1
if x > 0:
    print('x is positive')
```

```
x is positive
```

2.0.2 Syntax of a simple `if` statement: check one condition

```
if x > 0:
    print('x is positive')
```

- `if`: keyword for the Conditional Execution
- `x > 0`: the condition to check (logical/comparison operations)
  - if `True` (boolean value), the block following that condition `print('x is positive')` is executed
  - if `False` (boolean value), the block following that condition `print('x is positive')` is not executed
- colon `:`
- A new line
- indentation: 4 spaces

[7]:
```
if True:
    print("True")
```

```
True
```

[8]:
```
if False:
    print("True")
print("True")
```

```
True
```

[9]:
```
if False:
print("True")
```

```
  Cell In[9], line 2
    print("True")
    ^
```

```
IndentationError: expected an indented block after 'if' statement on line 1
```

[10]:
```
a = 1
b = 3

if a > b:
    print('a is bigger than b')
```

[11]:
```
a = 8
b = 3
c = 10

if a > b and c < b:
    print('a is bigger than b and c is smaller than b')
```

[12]:
```
a = 8
b = 3
c = 1

if a > b and c < b:
    print('a is bigger than b and c is smaller than b')
```

```
a is bigger than b and c is smaller than b
```

2.0.3 (2) If-else statement:

[13]:
```
x = 1
if x > 0:
    print('x is positive')
else:
    print('x is zero or negative')
```

```
x is positive
```

[14]:
```
x = 0
if x > 0:
    print('x is positive')
else:
    print('x is zero or negative')
```

```
x is zero or negative
```

[15]:
```
18 % 17
```

[15]: 1

% return the remainder of a division

[16]:
```
x = 1547
if x % 17 == 0:
    print('Your number is a multiple of 17.')
else:
    print('Your number is not a multiple of 17.')
```

```
Your number is a multiple of 17.
```

[17]:
```
x = int(input('Insert your number: '))
if x % 17 == 0:
    print('Your number is a multiple of 17.')
else:
    print('Your number is not a multiple of 17.')
```

```
Insert your number:
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[17], line 1
----> 1 x = int(input('Insert your number: '))
      2 if x % 17 == 0:
      3     print('Your number is a multiple of 17.')

ValueError: invalid literal for int() with base 10: ''
```

[18]:
```
# x = input('Insert your number: ')
x = 17.02
if x % 17 == 0:
    print('Your number is a multiple of 17.')
else:
    print('Your number is not a multiple of 17.')
```

```
Your number is not a multiple of 17.
```

2.0.4 Syntax of a `if-else` statement: check one condition and two potential executions

```
if x > 0:
    print('x is positive')
else:
    print('x is not positive')
```

- `if` and `else`: keywords for the Conditional Execution
- `x > 0`: the first condition
  - if `True`, the block following that condition is executed and the else statement is ignored.
  - if `False`, the block following the else statement is executed.

2.0.5 (3) If-elif-else statement:

[19]:
```
a = 3
b = 5

if a > b:
    print('a is bigger than b')
elif a < b:
    print('a is smaller than b')
else:
    print('a is equal to b')
```

```
a is smaller than b
```

[20]:
```
a = 3
b = 3

if a > b:
    print('a is bigger than b')
elif a < b:
    print('a is smaller than b')
else:
    print('a is equal to b')
```

```
a is equal to b
```

[21]:
```
a == b
```

[21]: True

[22]:
```
a = 3
b = 5

if a > b:
    print('a is bigger than b')
elif a > b:
    print('a is bigger than b')
else:
    print('a is not bigger than b')
```

```
a is not bigger than b
```

[23]:
```
a = 3
b = 5

if a > b:
    print('a is bigger than b')
elif a > b:
```

```
    print('a is bigger than b')
```

[24]:
```
a = 100
b = 3

if a > b:
    print('a is bigger than b')
elif a > b:
    print('a is bigger than b')
```

```
a is bigger than b
```

2.0.6 Syntax of a `if-elif-else` statement: check more than one conditions

```
if a > b:
    print('a is bigger than b')
elif a < b:
    print('a is smaller than b')
elif a < b:
    print('a is smaller than b')
elif a < b:
    print('a is smaller than b')
else:
    print('a is equal to b')
```

- `if`, `elif`, and `else`: keywords for the Conditional Execution
- `a > b`: the first condition
  - if `True`, the block following that condition is executed and rest is ignored.
  - if `False`, check the second condition after `elif`
- `a < b`: the second condition
  - if `True`, the block following that condition is executed and rest is ignored.
  - if `False`, the else statement is executed.

Conditions do not have to be mutually exclusive, but it makes better sense if they do!

2.0.7 Group Exercise

Using `if`, `elif` and `else` statements write a code where you check whether number a is larger than b and whether c is larger than d, and all the other potential relationships. For instance,

- if number a is larger than b and c is larger than d, the program print "a is larger than b and c is larger than d".
- if number a is smaller than b and c is smaller than d, the program print "a is smaller than b and c is smaller than d".
- if number a is larger than b and c is smaller than d, the program print "a is larger than b and c is smaller than d".
- if number a is smaller than b and c is larger than d, the program print "a is smaller than b and c is larger than d".
- if none of the above is true, the program print "a is equal to b or c is equal to d".

When you are done, raise your hand!

[26]:
```
a = 100
b = 5
c = 6
d = 7

if a > b and c>d:
    print("a is larger than b and c is larger than d")
elif a <b and c<d:
    print("a is smaller than b and c is smaller than d")
elif a > b and c<d:
    print("a is larger than b and c is smaller than d")
elif a < b and c>d:
    print("a is smaller than b and c is larger than d")
else:
    print("a is equal to b or c is equal to d")
```

```
a is larger than b and c is smaller than d
```

2.0.8 *Translate that!*

What does a `if-elif-else` statement do?

3 Standard Data Types in Python - strings

Category of Data type	Data type	Example
Numeric, scalar	Integer	1
	Floats	1.2
	Complex	1.5+0.5j
	Booleans	True
Container	strings	"Hello World"
	List	[1, "Hello World"]
	Tuple	(1, "Hello World")
	Set	{1, "Hello World"}
	Dictionary	{1: "Hello World", 2: 100}

3.1 What is a string in python?

- A sequence of characters
- Characters are ordered
- Immutable: characters can't be changed once created

3.2 Creating a String

- Assignment statement with `=`
- Function `str()`

[27]:
```
s = "A string of words"
s
```

[27]:
```
'A string of words'
```

[28]:
```
type(s)
```

[28]:
```
str
```

[29]:
```
s1 = 'A string of words'
s1
```

[29]:
```
'A string of words'
```

[30]:
```
s == s1
```

[30]:
```
True
```

[31]:
```
s2 = 'A string of words"
s2
```

```
  Cell In[31], line 1
    s2 = 'A string of words"
         ^
SyntaxError: unterminated string literal (detected at line 1)
```

[32]:
```
x = 10**2
x
```

[32]:
```
100
```

[33]:
```
xs = str(x)
xs
```

[33]:
```
'100'
```

[34]:
```
type(x)
```

[34]:
```
int
```

[35]:
```
type(xs)
```

[35]:
```
str
```

[36]:
```
int(xs)
```

[36]:
```
100
```

[40]:
```
a = "1000"
int(a)
```

```
1000
```

[41]:
```
a = 1000
type(a)
```

```
int
```

[42]:
```
a = 1000.0
type(a)
```

```
float
```

[43]:
```
a = '1000'
type(a)
```

```
str
```

[44]:
```
a = int("10100")
a
```

```
10100
```

[45]:
```
a = int("010100")
a
```

```
10100
```

[46]:
```
int("python")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[46], line 1
----> 1 int("python")

ValueError: invalid literal for int() with base 10: 'python'
```

[47]:
```
int("10.2")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[47], line 1
```

```
----> 1 int("10.2")

ValueError: invalid literal for int() with base 10: '10.2'
```

3.3 String concatenation

"addition" of two strings (with +)

[48]:
```
1 + 2
```

[48]:
```
3
```

[49]:
```
str_1 = 'hello'
str_2 = 'world'
```

[50]:
```
str_1 + str_2
```

[50]:
```
'helloworld'
```

[51]:
```
new_string = str_1 + str_2
new_string
```

[51]:
```
'helloworld'
```

Add a space (string ' ') in the middle of the two variables. A space is a character!

[52]:
```
a = str_1 + ' ' + str_2 + " "
a
```

[52]:
```
'hello world '
```

[53]:
```
a
```

[53]:
```
'hello world '
```

[54]:
```
2 *3
```

[54]:
```
6
```

[55]:
```
2 * a
```

[55]:
```
'hello world hello world '
```

[56]:
```
5 * a
```

[56]:
```
'hello world hello world hello world hello world hello world '
```

[57]:
```
str_1 * str_2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[57], line 1
----> 1 str_1 * str_2

TypeError: can't multiply sequence by non-int of type 'str'
```

[58]:
```
my_string = "hello world"
```

3.3.1 Group Exercise:

Create a new string variable `final_string` that adds three exclamation points to the end of `my_string`.

```
my_string = "hello world"
```

When you are done, raise your hand!

[59]:
```
my_string = my_string + "!!!"
my_string
```

[59]:
```
'hello world!!!'
```

3.4 Indexing String

To access each separate character in a string

Structure: `string[index]` * string variable name * square brackets * index: integer (starts from 0 in python)

[60]:
```
my_string = "Hello World"
```

[61]:
```
my_string[0]
```

[61]:
```
'H'
```

[62]:
```
my_string[4]
```

[62]:
```
'o'
```

Group Task: How do we get the last character in this string?

```
my_string = "Hello World"
```

When you are done, raise your hand

[63]:
```
my_string[10]
```

[63]:
```
'd'
```

[64]:
```
len(my_string)
```

[64]:
```
11
```

`len` is a built-in function that returns the number of characters in a string

[65]:
```
length = len(my_string)
length
```

[65]:
```
11
```

[66]:
```
my_string[length - 1]
```

[66]:
```
'd'
```

[67]:
```
my_string[len(my_string) -1 ]
```

[67]:
```
'd'
```

[68]:
```
my_string[11 -1]
```

[68]:
```
'd'
```

[69]:
```
my_string[11]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[69], line 1
----> 1 my_string[11]

IndexError: string index out of range
```

3.4.1 Access the last charater in a string

- Find the index of the last charater
  - A built-in function called len() that gives the information about length of an object
- Use that index to access the character

Python starts counting at zero!

The index of the last element will always be: len(string) - 1

3.4.2 Negative index

Another way to grab the last element so we don't need to calculate the length and substract one.

Count backwards!

[70]:
```
my_string
```

[70]:
```
'Hello World'
```

[71]:
```
my_string[-1]
```

[71]:
```
'd'
```

[72]:
```
my_string[-2]
```

[72]:
```
'l'
```

3.5 Assignment: HW2

- released today
- due by 09/18

4 Next Class

- String
- Iterations

Readings: Chapter 7