Branch: master ▾ Find file Copy path

ma591spring2020 / material / loopsconditionals.ipynb

 asaibab Add files via upload

9f11f9a on Feb 14

1 contributor

 Raw Blame History   

634 lines (633 sloc) 12.8 KB

Loops and conditionals

We will learn

1. `if/elif/else` statements
2. `while` loops
3. `for` loops

if/elif/else

```
"If you can talk with crowds and keep your virtue,
    Or walk with Kings—nor lose the common touch,
If neither foes nor loving friends can hurt you,
    If all men count with you, but none too much;
If you can fill the unforgiving minutem
    With sixty seconds' worth of distance run,
Yours is the Earth and everything that's in it,
    And—which is more—you'll be a Man, my son!"
                    - Rudyard Kipling
```

If/Else

The general structure of an if/else test is

```
if condition: #Note the colon
    <a set of statements executed if condition is True>
else:         #Note the colon
    <a set of statements executed if condition is False>
```

This is useful for functions defined piecewise. Example

$$H(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x<0. \end{cases}$$

In [1]:
```
x = 5.
if x >= 0.:
    H = 1.
else:
    H = 0.
print(H)
```

```
1.0
```

Inline If statement

The conditional

```
if condition:
```

```
if condition:
    a = v1
else:
    a = v2
```

can be written in an alternative fashion

```
a = v1 if condition else v2
```

For the Heaviside function:

In [2]:
```
H = 1. if x >= 0. else 0.
print(H)
```

```
1.0
```

if/elif/else

Consider the sign function

$$sign(x) = \begin{cases} 1 & x>0 \\ 0 & x = 0 \\ -1 & x<0 \end{cases}$$

In [3]:
```
x = 2

if x > 0:
    sign = 1
elif x == 0:
    sign = 0
elif x < 0:
    sign = -1
else:
    print("Should never reach here")

print('The sign of x = %g is %d.' %(x, sign) )
```

```
The sign of x = 2 is 1.
```

While

```
I look at the world and I notice it's turning
While my guitar gently weeps
With every mistake we must surely be learning
Still my guitar gently weeps
                        - George Harrison
```

While loops

Syntax:

```
while <condition not met>:
    <do something>
    <update condition>
```

Warning: be careful of infinite loops.

Example: Count all multiples of 3 less than 20.

In [4]:
```
num = 1
count = 0
while num < 20:
    if num % 3 == 0:
        count += num
    num += 1

print('The sum of multiples of 3 less than 20 is ', count)
```

```
The sum of multiples of 3 less than 20 is  63
```

For

```
Therefore, send not to know
For whom the bell tolls,
It tolls for thee.
             - John Donne
```

For loops

- Allow the user to perform the same operation on every element of a list
- Preferable over while, especially if the number of iterations is known in advance

In [5]:
```
for i in range(10):
    print(i)
    if i%7 == 0:
        print(i, "is divible by 7.")
```

```
0
0 is divible by 7.
1
2
3
4
5
6
7
7 is divible by 7.
8
9
```

Details on range

- `range(n)` generates integers 0,1,...,n-1.
- `range(start,stop,step)` generates start, start + step, ..., ends before stop.

In [6]:
```
print(list(range(5)))
```

```
[0, 1, 2, 3, 4]
```

In [7]:
```
print(list(range(2,8,3)))
```

```
[2, 5]
```

In [8]:
```
print(type(range(2)))  # Range is an iterable object
```

```
<class 'range'>
```

Break and continue

Two other useful commands

- `break` breaks out of the loop
- `continue` breaks out of the current iteration, but continues the iterative process

These can be used in conjunction with `while` or `for` loops

In [9]:
```
## Break
for j in range(10):
    if j == 5:
        break
    print('The value of j is ', j)
```

```
The value of j is  0
The value of j is  1
The value of j is  2
The value of j is  3
The value of j is  4
```

In [10]:
```
## Continue
for j in range(10):
    if j == 5:
        continue
    print('The value of j is ', j)
```

```
The value of j is  0
The value of j is  1
The value of j is  2
The value of j is  3
The value of j is  4
The value of j is  6
The value of j is  7
The value of j is  8
The value of j is  9
```

Loops and lists

Can iterate through one or multiple lists in different ways

Can iterate through one or multiple lists in different ways

In [11]:
```
luminaries = ['Euler', 'Gauss', 'Ramanujam']
for person in luminaries:
    print('The mathematician is ', person)
```

```
The mathematician is  Euler
The mathematician is  Gauss
The mathematician is  Ramanujam
```

In [12]:
```
nationalities = ['Swiss', 'German', 'Indian']
for person, country in zip(luminaries,nationalities): # Iter
ate through multiple lists
    print("The mathematician ", person, " was ", country)
```

```
The mathematician  Euler  was  Swiss
The mathematician  Gauss  was  German
The mathematician  Ramanujam  was  Indian
```

Loops and lists

A **list comprehension** is useful in constructing a list using a for loop

```
lst = [function(e) for e in list]
```

In [13]:
```
lst = [j**2 for j in range(10)]

print(lst)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Iterating through lists but with a counter.

In [14]:
```
for j in range(len(luminaries)):
    print('Mathematician #', j, 'is ', luminaries[j])

#Alternative
for j, person in enumerate(luminaries): #Gives a counter ove
r a list
    print('Mathematician #', j, 'is ', person)
```

```
Mathematician # 0 is  Euler
Mathematician # 1 is  Gauss
Mathematician # 2 is  Ramanujam
Mathematician # 0 is  Euler
Mathematician # 1 is  Gauss
Mathematician # 2 is  Ramanujam
```

Loops and dictionaries

Syntax:

```
for key, value in dictionary.items():
```