CS-468 Artificial Intelligence - Lab Manual - 2

June 21, 2021

1 Type Conversion

Type conversion or type casting is the process of converting the value of one type to another type. Python has two types of type conversion:

1. Implicit Type Conversion
2. Explicit Type Conversion

1.1 Implicit Type Conversion

In implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

Given below is an example where Python performs conversion of the lower data type (integer) to the higher data type (float) to avoid data loss.

[1]:
```python
num_int = 33
num_float = 23.3

new_num = num_int + num_float

print("Data type of num_int: ", type(num_int))
print("Data type of num_float: ", type(num_float))

print("Value of new_num = ", new_num)
print("Data type of new_num: ", type(new_num))
```

```
Data type of num_int:  <class 'int'>
Data type of num_float:  <class 'float'>
Value of new_num =  56.3
Data type of new_num:  <class 'float'>
```

Note: Any operation that involves both integers and floating point numbers will always produce a floating point number. That is, if one of the sides of an operation such as add, subtract, divide or multiple is a floating point number then the result will be a floating point number

Now, let's try adding a string and an integer and see how Python deals it.

[2]:
```python
num_int = 55
num_string = "655"
```

```
print("Data type of num_int: ", type(num_int))
print("Data type of num_string: ", type(num_string))

print(num_int + num_string)
```

```
Data type of num_int:  <class 'int'>
Data type of num_string:  <class 'str'>
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-8cc362128594> in <module>
      5 print("Data type of num_string: ", type(num_string))
      6
----> 7 print(num_int + num_string)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

1.2 Explicit Type Conversion

In explicit type conversion, the programmer converts the data type of an object to required data type. To achieve this, Python provides various built-in functions like `int()`, `float()`, `bool()`, `str()`, etc.

1.2.1 Converting to Ints

It is possible to convert another type into an integer using the `int()` function. For example, if we want to convert a string into an int (assuming the string contains a integer number) then we can do this using the `int()` function.

This can be useful when used with the `input()` function because the `input()` always returns a string.

[3]:
```
# Consider a string containing an integer value
str_int = "55"

# Print type of str_int
print("Data type of str_int: ", type(str_int))

# Convert type of str to int
int_val = int(str_int)

# Print type of int_val
print("Data type of int_val: ", type(int_val))
```

```
Data type of str_int:  <class 'str'>
Data type of int_val:  <class 'int'>
```

[4]:
```
# Ask the user to input age
age = input("Enter your age: ")

# Short hand
# age = int(input("Enter your age: "))

# Print type of age
print("Data type of age: ", type(age))

# Type cast to int
age = int(age)


# Print type of age
print("Data type of age: ", type(age))
```

```
Enter your age: 22
Data type of age:  <class 'str'>
Data type of age:  <class 'int'>
```

1.3 Converting to Floats

As with integers, it is also possible to convert to other types such as int to float, string to float etc. This is done using `float()` function.

[5]:
```
# define two variables
int_value = 2
string_value = "1.5"

# Converting int to float
float_value = float(int_value)

# Print types
print("Data type of int_value: ", type(int_value))
print("Data type of string_value: ", type(string_value))
print("Data type of float_value after converting int_value to float: ",␣
 ↪type(float_value))
print("Float_Value = ", float_value)
```

```
Data type of int_value:  <class 'int'>
Data type of string_value:  <class 'str'>
Data type of float_value after converting int_value to float:  <class 'float'>
Float_Value =  2.0
```

Similarly, you can convert the value to float when using `input()` function.

[6]:
```
# Take input mass and convert to float
mass = float(input("Enter mass: "))
```

```
# Take input acceleration and conver to float
acceleration = float(input("Enter acceleration: "))

# Calculate Force
force = mass * acceleration

print("Force = ", force)
```

```
Enter mass: 3
Enter acceleration: 3.3
Force =  9.899999999999999
```

1.4 Converting to Strings

You can convert values to string as well. For this, `str()` function is used.

[7]:
```
# Define two variables
num = 5
float_num = 3.14137

# Print data type
print("Data type of num: ", type(num))
print("Data type of float_num: ", type(float_num))

# Convert to string
string_num = str(num)
string_float_num = str(float_num)

# Print data type
print("Data type of string_num: ", type(string_num))
print("Data type of float_num: ", type(string_float_num))
```

```
Data type of num:  <class 'int'>
Data type of float_num:  <class 'float'>
Data type of string_num:  <class 'str'>
Data type of float_num:  <class 'str'>
```

1.5 Type Conversion Table

Following are some of the most commonly used functions for type conversion:

Function	Description
`int(item)`	You pass an argument to the int() function and it returns the argument's value converted to an int
`float(item)`	You pass an argument to the float() function and it returns the argument's value converted to a float
`str(item)`	You pass an argument to the str() function and it returns the argument's value converted to a str

Function	Description
`bool(item)`	You pass an argument to the bool() function and it returns the argument's value converted to a bool.

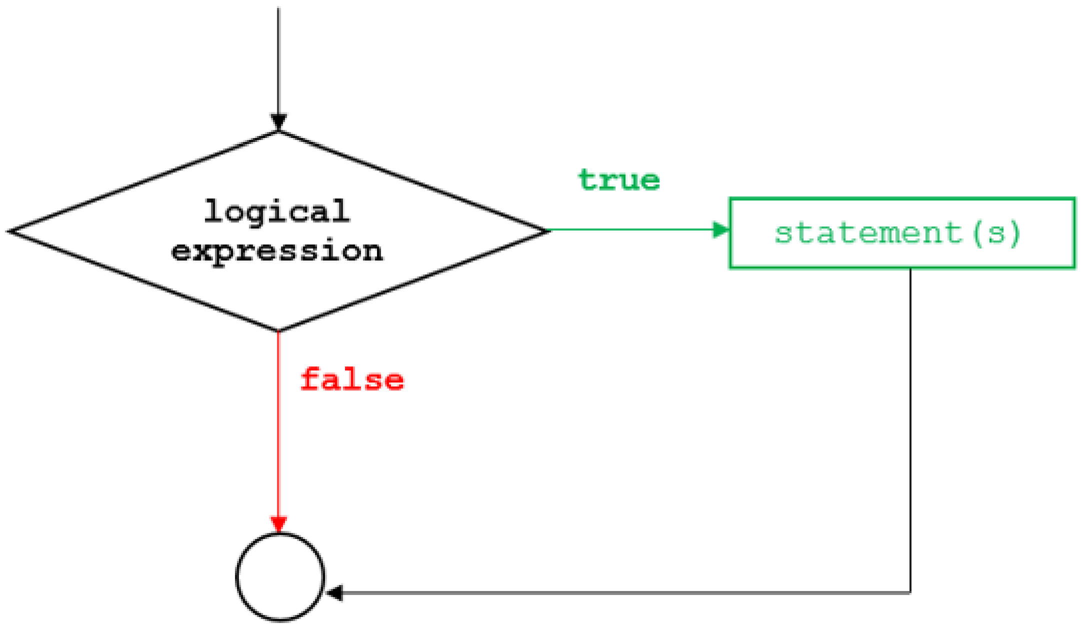
Note: Non-zero numeric values are converted to True and 0 is converted to False

2 Conditional Statements: if, else, elif

Often, we want to execute the code to perform some action based on a condition. For example, if the condition is true, do something. We can get this done using **"if, else, elif"** statements. Python's **"if"** statement select actions to perform.

2.1 if Statement

- The **if** statement used to create a single-way decision structure
- The **if** statement causes one or more statements to execute only when a boolean expression is true



Let's learn with the help of examples.

```
[8]: # Let say, we have a conditon
     if 3 > 2:
         print ('True')
     # print statement will only execute if the condition is satisfied
     # Notice, with print statement, we will not see typical output cell␣
      ↪[some_number]
```

```
True
```

2.1.1 Use of Indentation

Notice *a block of whitespace before the print( ) statement, in the cell above. It is very important in Python. Python does not use brackets in order to separate a block of code in the execution statements, it uses white spaces instead. Like most of the other IDEs, jupyter notebook automatically does this indentation of Python code after a colon.*

If the condition is not satisfied (in the example below), print statement will not be executed. Nothing will appear in the output.

[9]:
```
if 3 < 2:
    print ('True')
```

So, in the above cell, the condition is not satisfied and we did not get True in the output. Well, do we really know if the code was executed!

[10]:
```
x = 7
y = 5

if x > y:
    print("x is smaller")
    print("I am still inside if statement")

```

```
x is smaller
I am still inside if statement
```

[11]:
```
x = 7
y = 5

if x < y:
    print("x is smaller")
print("Control is outside if")
```

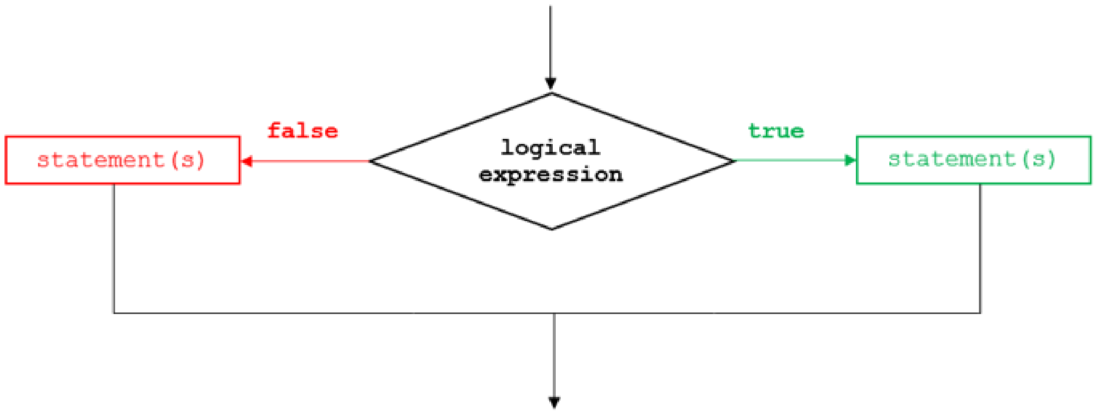
```
Control is outside if
```

In such situation, we usually want our program to do something, even if the condition is not satisfied, e.g., print False.

This is where we (can) introduce **else**.

2.2 if-else Statement

- if else statement provides *Two-way selection*
- The if-else statement will execute one group of statements if its Boolean expression is true, or another group if its boolean expression is false



```
[12]: if 3 == 2: # You can change == to != to print True
          print ('True')
      else:
          print ('False')
```

```
False
```

```
[13]: # Ask the user to input a number
      num = int(input("Enter a number: "))
      if num % 2 == 0:
          print("Even")
      else:
          print("Odd")
```

```
Enter a number: 3
Odd
```

For example, if we wish to decide if someone is a teenager or not then we might check to see if they are over 12 and under 20.

```
[14]: age = 15
      status = None
      if (age > 12) and (age < 20):
       status = 'teenager'
      else:
       status = 'not teenager'
      print(status)
```

```
teenager
```

Note: *None* keyword is used to define null variable or an object in Python

2.3 if elif (Multiple Conditions)

We can have multiple conditions in the code! Let's introduce elif in such situation. * (elif is a short of else-if and we can introduce many elif statements in the code.) *

[15]:
```python
if 1 == 2:
    print('1st statement')
elif 2 == 2:
    print('2nd statement')
else:
    print('3rd statement')
```

```
2nd statement
```

** *Very important thing to remember for elif statement.* ** The code in the *"if-elif(s)-else"* block will only execute the first True condition, it does not matter if you have other True conditions in the code after your first True.

Let's try another example here!

[16]:
```python
if 1 == 2:
    print('1st statement')
elif 3 == 3:
    print('2nd statement')
elif 2 == 2:
    print('3rd statement')
else:
    print('4rd statement')
```

```
2nd statement
```

In the code above, although 2==2 is True, however, print('3rd statement') will not be executed. Since, the condition for print('2nd statement') is also True and have already been executed!

2.4 Nesting if Statements

It is possible to *nest* one `if` statement inside another. This term nesting indicates that one if statement is located within part of the another if statement and can be used to refine the conditional behaviour of the program.

[17]:
```python
is_snowing = True
temp = -1
if temp < 0:
    print("It's freezing")
    if is_snowing:
        print("Put on your boots")
    print("Time for a cup of tea")
print("Bye")
```

```
It's freezing
Put on your boots
Time for a cup of tea
Bye
```

2.4.1 A note on True and False

Python is actually quite flexible when it comes to what actually is used to represent `True` and `False`, in fact the following rules apply

- 0, "" (empty strings), `None` equate to `False`
- Non zero, non-empty strings, any object equate to `True`

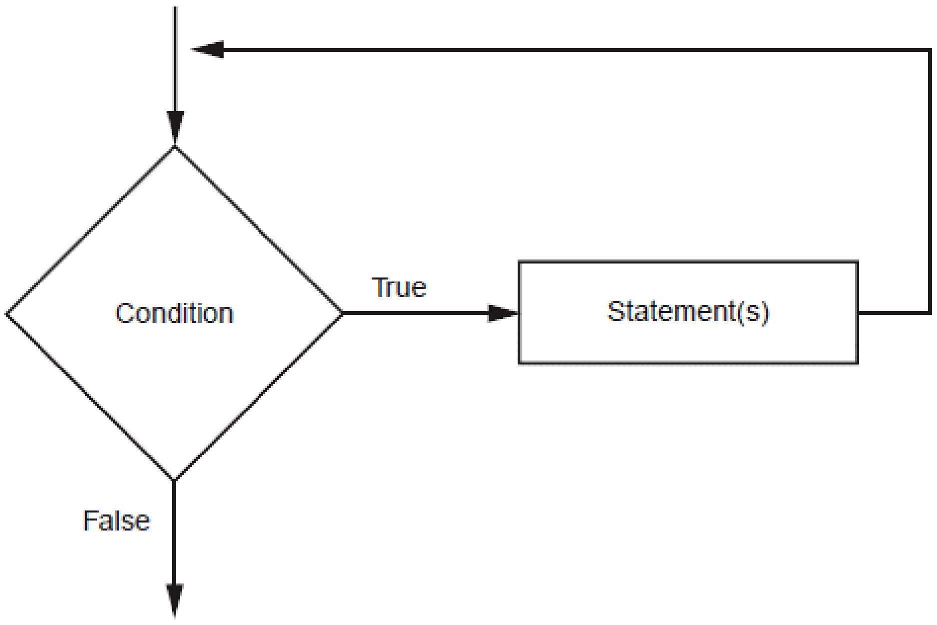
It is recommended sticking to just `True` and `False` as it is cleaner and safer approach

3 Loops

- A loop structure causes a statement or a set of statements to execute repeatedly based on a condition
- The `while` and `for` are the two main loops in Python for this purpose

3.1 while loop

while loop continually perform an action, until some condition has been met. Before executing the statement(s), that are nested in the body of the loop, Python keeps evaluating the test expression (loop test), until the test returns a False value.



Let's start with a simple while loop in the cell below.

[18]:
```
# A simple while loop
i = 1 # initializing a variable i = 1
while i < 5: # loop test
    # Run the block of code (given below), till "i < 5"
    print('The value of i is: {}'.format(i))
    i = i+1 # increase i by one for each iteration
```

```
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
```

What if we don't have i = i+1 statement? Well, we will get into an infinite while loop, because i will alway be less than 5, right? This is not what we want…….!

Note: If you get into an infinite while loop, you will notice a continuous out put or asterisk on left side of your cell in the jupyter notebook for a very long time **"In [*]"** . Go to the Kernel and restart to get out of this situation

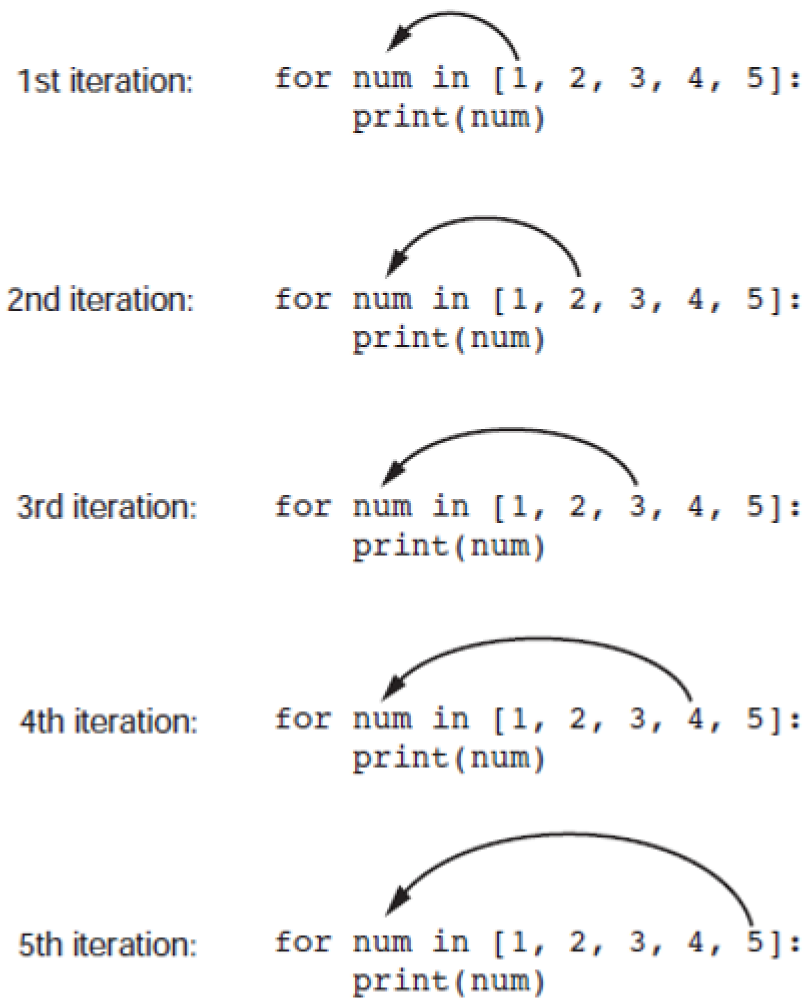
A nice and cleaner way of exiting the while loop is using else statement (optional):

[19]:
```
i = 1
while i < 5:
    print('The value of i is: {}'.format(i))
    i = i+1
else:
    print('Exit loop')
```

```
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
Exit loop
```

4 for Loop

- In many cases we know how many times we want to iterative over one or more statements.
- A count-controlled loop iterates a specific number of times. In Python you use the for statement to write a count-controlled loop
- The for loop allows us to iterate through a sequence
- The for loop works on strings, lists, tuples, and built-in iterables, as well as on new user-defined objects



```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

Let's create a list "my_list" of numbers and run a "for" loop using that list.

[20]:
```
my_list = [1,2,3,4,5]
```

In a very simple situation, we can execute some block of code using "for" loop for every single item in "my_list", let's try this out!

[21]:
```
for item in my_list:
    print(item)
```

```
1
2
3
4
5
```

Note: In the "for" loop above, the temporary variable "item" can be whatever you want e.g. i, a, x, name, num etc. A good practice is to use it carefully so that you can be self-explanatory and easier to remember. A good selection in this case is "num", because we have numbers in my_list.

TIP: A very useful trick is, always use comments in your code, this is a great way to remember and also helpful while working in a team.

[22]:
```python
for num in my_list:
    print('Hello world')
```

```
Hello world
Hello world
Hello world
Hello world
Hello world
```

Did you notice something in the above cell? I have replaced "item" with "num" and also the block of code in the print statement. So, the print('Hello world') does not need to be related to items in the sequence (my_list in this case), we can print anything, we want!

Let's print the square of the numbers in my_list using for loop

[23]:
```python
# printing square of the numbers in myList.
for num in my_list:
    print(num**2)
```

```
1
4
9
16
25
```

This was all about while and for loops at the moment!

A quick reminder: While working in jupyter notebook, <shift + tab> is always useful to expore the document string of the related method.

After discussing loops, its time to introduce a very useful loop-related function range( ).

range():

Rather than creating a sequence (specially in for loops), we can use range(), which is a generator of numerical values. * With one argument, range generates a list of integers from zero up to, but not including, the argument's value. * With two argument, the first is taken as the lower bound (start). * We can give a third argument as a step, which is optional. If the third argument is provided,Python adds the step to each successive integer in result (default value of step is "+1").

Some working examples of range are given in the below code cels.

[24]:
```python
# This (the code below) will give the range object, which is iterable.
range(5)
```

[24]:
```
range(0, 5)
```

[25]:
```python
# With method "list", we can convert the range object into a list
list(range(5))
```

```
[25]: [0, 1, 2, 3, 4]
```

```
[26]: # Use of range(), with single argument, in a loop
      for i in range(5):
          print(i)
```

```
0
1
2
3
4
```

```
[27]: # Use of range(), with two argument, in a loop
      for i in range(3,5):
          print(i)
```

```
3
4
```

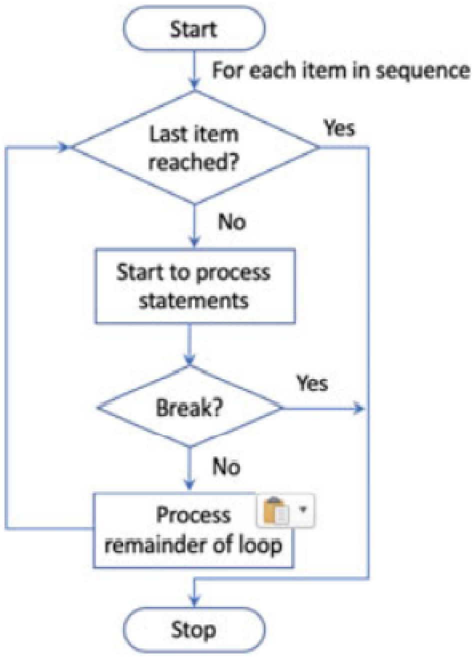
```
[28]: # Use of range(), with three argument, in a loop
      for i in range(1,10,2):
          print(i)
```

```
1
3
5
7
9
```

5 Break Loop Statement

Python allows programmers to decide whether they want to break out of a loop early or not (whether we are using a for loop or a while loop). This is done using the `break` statement.

The `break` statement allows a developer to alter the normal cycle of the loop based on some criteria which may not be predictable in advance (for example it may be based on some user input).


Start
For each item in sequence
Last item reached?
Yes
No
Start to process statements
Break?
Yes
No
Process remainder of loop
Stop

The `break` statement, when executed, will terminate the current loop and jump the program to the first line after the loop

Typically, a guard statement (if statement) is placed on the break so that the break statement is conditionally applied when appropriate.

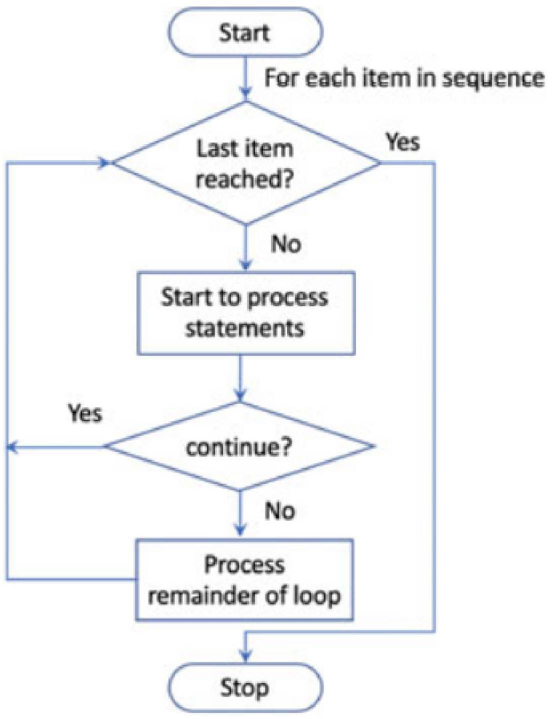
This is shown below for a simple application where the user is asked to enter a number which may or may not be in the range defined by the for loop. If the number is in the range, then we will loop until this value is reached and then break the loop (that is terminate early without processing the rest of the values in the loop):

[29]:
```python
print('Only print code if all iterations completed')
num = int(input('Enter a number to check for: '))
for i in range(0, 6):
    if i == num:
        break
    print(i, ' ')
print('Done')
```

```
Only print code if all iterations completed
Enter a number to check for: 3
0
1
2
Done
```

6 Continue the Loop

The continue statement also affects the flow of control within the looping constructs for and while. However, it does not terminate the whole loop; rather it only terminates the current iteration round the loop. This allows you to skip over part of a loop's iteration for a particular value, but then to continue with the remaining values in the sequence.


Start
For each item in sequence
Last item reached?
Yes
No
Start to process statements
Yes
continue?
No
Process remainder of loop
Stop

A guard (if statement) can be used to determine when the continue statement should be executed.

As with the break statement, the continue statement can come anywhere within the body of the looping construct. This means that you can have some statements that will be executed for every value in the sequence and some that are only executed when the continue statement is not run.

This is shown below. In this program the continue statement is executed only for odd numbers and thus the two print() statements are only run if the value of i is even:

[30]:
```python
for i in range(0, 10):
    print(i, ' ', end='')
    if i % 2 == 1:
        continue
    print('hey its an even number')
    print('even numbers are easy to manipulate')
print('Done')
```

```
0  hey its an even number
even numbers are easy to manipulate
1  2  hey its an even number
```

```
even numbers are easy to manipulate
3  4  hey its an even number
even numbers are easy to manipulate
5  6  hey its an even number
even numbers are easy to manipulate
7  8  hey its an even number
even numbers are easy to manipulate
9  Done
```

7 Exercise

7.1 Mass and Weight

Scientists measure an object's mass in kilograms and its weight in newtons. If you know the amount of mass of an object in kilograms, you can calculate its weight in newtons with the following formula: **

weight = mass * 3 9.8

** Write a program that asks the user to enter an object's mass, and then calculates its weight. If the object weighs more than 500 newtons, display a message indicating that it is too heavy. If the object weighs less than 100 newtons, display a message indicating that it is too light.

7.2 Hot Dogs Cookout Calculator

Assume that hot dogs come in packages of 10, and hot dog buns come in packages of 8. Write a program that calculates the number of packages of hot dogs and the number of packages of hot dog buns needed for a cookout, with the minimum amount of leftovers. The program should ask the user for the number of people attending the cookout and the number of hot dogs each person will be given. The program should display the following details: * The minimum number of packages of hot dogs required * The minimum number of packages of hot dog buns required * The number of hot dogs that will be left over * The number of hot dog buns that will be left over

7.3 Miles Per Gallon

Drivers are concerned with the mileage obtained by their automobiles. One driver has kept track of several tankfuls of gasoline by recording miles driven and gallons used for each tankful. Develop a sentinel-controlled-repetition script that prompts the user to input the miles driven and gallons used for each tankful. The script should calculate and display the miles per gallon obtained for each tankful. After processing all input information, the script should calculate and display the combined miles per gallon obtained for all tankfuls (that is, total miles driven divided by total gallons used).

- Enter the gallons used (-1 to end): 12.8
- Enter the miles driven: 287
- The miles/gallon for this tank was 22.421875
- Enter the gallons used (-1 to end): 10.3
- Enter the miles driven: 200
- The miles/gallon for this tank was 19.417475
- Enter the gallons used (-1 to end): 5

- Enter the miles driven: 120
- The miles/gallon for this tank was 24.000000
- Enter the gallons used (-1 to end): -1
- The overall average miles/gallon was 21.601423

7.4 Software Sales

A software company sells a package that retails for $99. Quantity discounts are given according to the following table:

Quantity	Discount
10–19	10%
20–49	20%
50–99	30%
100 or more	40%

Write a program that asks the user to enter the number of packages purchased. The program should then display the amount of the discount (if any) and the total amount of the purchase after the discount.

7.5 Shipping Charges

The Fast Freight Shipping Company charges the following rates:

Weight of Package	Rate per Pound (Dollar)
2 pounds or less	1.50
Over 2 pounds but not more than 6 pounds	3.00
Over 6 pounds but not more than 10 pounds	4.00
Over 10 pounds	4.75

Write a program that asks the user to enter the weight of a package and then displays the shipping charges.

7.6 Body Mass Index

Write a program that calculates and displays a person's body mass index (BMI). The BMI is often used to determine whether a person is overweight or underweight for his or her height. A person's BMI is calculated with the following formula:

**

BMI = weight * 703 / height^2

**

where weight is measured in pounds and height is measured in inches. The program should ask the user to enter his or her weight and height and then display the user's BMI. The program should also display a message indicating whether the person has optimal weight, is underweight, or is overweight. A person's weight is considered to be optimal if his or her BMI is between 18.5 and 25. If the BMI is less than 18.5, the person is considered to be underweight. If the BMI value is greater than 25, the person is considered to be overweight.

8 Exercise (Loops)

8.1 Budget Analysis

Write a program that asks the user to enter the amount that he or she has budgeted for a month. A loop should then prompt the user to enter each of his or her expenses for the month and keep a running total. When the loop finishes, the program should display the amount that the user is over or under budget.

8.2 Average Rainfall

Write a program that uses nested loops to collect data and calculate the average rainfall over a period of years. The program should first ask for the number of years. The outer loop will iterate once for each year. The inner loop will iterate twelve times, once for each month. Each iteration of the inner loop will ask the user for the inches of rainfall for that month. After all iterations, the program should display the number of months, the total inches of rainfall, and the average rainfall per month for the entire period.

8.3 Guess the Number

Write a script that plays "guess the number." Choose the number to be guessed by selecting a random integer in the range 1 to 1000. Do not reveal this number to the user. **Display the prompt "Guess my number between 1 and 1000 with the fewest guesses:"**. The player inputs a first guess. If the guess is incorrect, display **"Too high. Try again."** or **"Too low. Try again."** as appropriate to help the player "zero in" on the correct answer, then prompt the user for the next guess. When the user enters the correct answer, display **"Congratulations. You guessed the number!"**, and allow the user to choose whether to play again.

8.4 Population

Write a program that predicts the approximate size of a population of organisms. The application should allow the user to enter the starting number of organisms, the average daily population increase (as a percentage), and the number of days the organisms will be left to multiply. For example, assume the user enters the following values:

Starting number of organisms: 2

Average daily increase: 30%

Number of days to multiply: 10

The program should display the following table of data:

Day Approximate	Population
1	2
2	2.6

Day Approximate	Population
3	3.38
4	4.394
5	5.7122
6	7.42586
7	9.653619
8	12.5497
9	16.31462
10	21.209

[ ]:

```

```