

Introduction to Computer Programming

Week 10.1: Review



Structure of session

- To provide an overview of **core** Python functionality and programming techniques
- Not exhaustive - so please see the lecture slides from Weeks 1-7 for more details
- Short exercises with solutions presented
- **Please open the IDE of your choice**

Mathematical operations

| Operation | Description | Example |
|-----------|---|------------------|
| + | Addition | 5 + 3 = 8 |
| - | Substraction | 5 - 3 = 2 |
| * | Multiplication | 5 * 3 = 15 |
| / | Division | 5 / 3 = 1.666... |
| // | Floor division (round down to an integer) | 5 // 3 = 1 |
| % | Modulo (compute remainder) | 5 % 3 = 2 |
| ** | Exponent | 5 ** 3 = 125 |

Boolean operations

| Operation | Description | Example | Value |
|-----------|---------------------------|---------|-------|
| == | Is equal? | 1 == 2 | False |
| != | Is not equal? | 1 != 2 | True |
| < | Less than? | 1 < 2 | True |
| > | Greater than? | 1 > 2 | False |
| <= | Less than or equal to? | 1 <= 2 | True |
| >= | Greater than or equal to? | 1 >= 2 | False |

Logical operations

| Operation | Description | Example | Value |
|-----------|------------------------|-----------------|-------|
| and | Are both true? | 1 < 2 and 3 < 2 | False |
| or | Is one true? | 1 < 2 or 3 < 2 | True |
| not | Negate the conditional | not(1 < 2) | False |

Exercise:

Use logical operations to determine whether an integer N is a multiple of four. Can you also determine whether N is an odd multiple of four?

In [] :

Basic variable types

- **Ints:** integers; e.g. a = 2
- **Floats:** floating-point numbers with decimals; e.g. a = 2.0
- **Strings:** collection of characters contained in single or double quotes; individual characters can be accessed using an index (starting at 0)

In [2] :

```
s = 'Hello'
print(s[1])
e
```

Use the `int`, `float`, and `str` functions to convert between types

In [3] :

```
a = 2.0
print(int(a))
2
```

Data structures

| Type | Example | Characteristics |
|------------|---------------------------|--------------------------------------|
| List | L = [1, 1.0, 'one'] | Mutable, iterable, ordered |
| Tuple | t = (1, 1.0, 'one') | Immutable, iterable, ordered |
| Set | s = {1, 1.0, 'one'} | Mutable, iterable, unordered, unique |
| Dictionary | d = {'a':1, 'b':2, 'c':3} | Mutable, iterable, ordered |

- **Mutable:** Can be modified
- **Immutable:** Cannot be modified
- **Ordered:** Elements can be accessed using an index or a key

Data structures continued

- Use `list`, `tuple`, and `set` functions to convert between types
- Elements in lists and tuples can be accessed using an integer index (starting at 0)
- Elements in dictionaries are accessed using keys

In [4] :

```
l = [1, 2, 3, 3]
print(set(l)) # convert a list to a set
print(l[0]) # accessing the first entry of the list l
{1, 2, 3}
```

In [5] :

```
# create a dict of gravitational accelerations
g = {'Earth': 9.8, 'Mars':3.7, 'Jupiter':25}
print(g['Earth'])
9.8
```

If statements

- Used to make a decision in a program
- Runs an indented block of code if a conditional statement is true

In [48] :

```
i = 5
if i < 10:
    print("Doing something because i < 10")
print('Printing non-indented code for all values of i')
Doing something because i < 10
Printing non-indented code for all values of i
```

If-else statements

- Creates two pathways, the choice depends on whether a condition is true or false

In [50] :

```
i = 5
if i < 10:
    print('Doing something')
else:
    print('Doing something else')
Doing something
```

If-else-elif statements

- Creates multiple pathways, the choice depends on which condition is true

In [51] :

```
i = 20
if i < 10:
    print('Doing something')
elif i > 10:
    print('Doing something else')
else:
    print('Doing something different from the other two cases')
Doing something else
```

Exercise

- A currency converter will change UK pounds into Canadian dollars using the formula
$$C = rP$$
where r is the conversion rate, P is the number of pounds, and C is the amount of Canadian dollars.
- The rate r depends on the number of pounds P being converted:

| Pounds P | Rate r |
|--|----------|
| under £5,000 | 1.64 |
| between £5,000 and £10,000 (inclusive) | 1.66 |
| over £10,000 | 1.70 |

- Write a program to calculate the Canadian pounds C that will be converted for a given number of UK pounds P

Solution

In [] :

For loops

- For repeating code a fixed number of times

```
for e in collection:
    # run indented code
```

- The indented code is run until `e` has taken on every value in `collection` (which is an iterable object like a list or tuple)

In [10] :

```
for i in range(5):
    print(i, end=" ")
0 1 2 3 4
```

In [11] :

```
for c in ['red', 'blue', 'green']:
    print(c.capitalize(), end=" ")
Red, Blue, Green,
```

While loops

- For repeating code until a condition becomes false

```
while condition:
    # run indented code
```

- While loops are useful when you don't know how many times to repeat code
- Beware of infinite loops!

In [1] :

```
# compute the square numbers that are smaller than 450
i = 1
while i**2 < 450:
    print(i**2, end=" ")
    i += 1
1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441,
```

Break and continue

- `break` is used to terminate a loop
- `continue` is used to skip an iteration in a loop

In [13] :

```
for i in range(10):
    print(i, end=" ")
0 1 2 3 4 5 6 7 8 9
```

In [14] :

```
for i in range(10):
    if i == 4:
        break
    print(i, end=" ")
0 1 2 3
```

In [15] :

```
for i in range(10):
    if i == 4:
        continue
    print(i, end=" ")
0 1 2 3 5 6 7 8 9
```

Exercise

1. Write a program that calculates how many letters appear before the first e in a word. For example, in the word "programmer", there are 8 letters before the first e.
2. What would you change in your program if you wanted to count all the letters except e?

In [] :

Changing `break` to `continue` counts all of the letters except e

Functions

- **Functions** are mini-programs based on a collection of code that has been given a name
- Functions are defined using the `def` keyword
- Function inputs are called **arguments**
- The `return` keyword is used to output data from a function

In [17] :

```
# add two numbers a and b together
def my_sum(a, b):
    c = a + b
    return c
c = my_sum(3, 6)
print(c)
9
```

The unpacking operator

- The unpacking operator `*` is used to define functions with an arbitrary number of arguments

In [3] :

```
# sums an arbitrary number of numbers
def my_sum(*numbers):
    s = 0
    for n in numbers:
        s += n
    return s
S = my_sum(1, 2, 3, 4, 5)
print(S)
15
```

Keyword arguments

- Keyword arguments allow arguments to be provided in any order

In [4] :

```
# create a function to display someone's name
def print_name(first_name, second_name):
    print('The name is', first_name, second_name)
# using standard (positional) arguments: order matters
print_name('Isaac', 'Newton')
# using keyword arguments: order does not matter
print_name(second_name = 'Newton', first_name = 'Isaac')
The name is Isaac Newton
The name is Isaac Newton
```

Default arguments

- Default arguments pre-assigns a value to optional arguments
- Default values are assigned in the function definition
- Default arguments must be the last arguments in a function

In [6] :

```
# this function divides two numbers (n = numerator and d = denominator)
def my_divide(n, d = 1):
    print(n / d)
my_divide(5)
my_divide(3, 4)
5.0
0.75
```

Variable scope

- **Local** variables can only be accessed within the functions that create them

In [21] :

```
def my_sum(x, y):
    # create a local variable z
    z = x + y
my_sum(2, 5)
# attempt to access a local variable
print(z)
```

```
.....
NameError                                Traceback (most recent call last)
<ipython-input-21-83ff9cb5c933> in <module>
      6
      7 # attempt to access a local variable
----> 8 print(z)
NameError: name 'z' is not defined
```

Variable scope

- **Global** variables can be accessed anywhere (**but should be avoided**)
- Variables defined in the main python code are global variables
- The `global` keyword is used to convert local variables into global variables

In [22] :

```
x = 4
def print_x():
    print(x)
print_x()
4
```

In [23] :

```
def my_sum(x, y):
    global z
    z = x + y
my_sum(2, 5)
print(z)
7
```

Exercise

Write a function that computes the potential energy of an object using the equation $E = mgh$. The function should take as inputs:

- m , mass in kg
- g , the gravitational acceleration in m/s^2
- h , height in m

Follow up: how would you change your code to set $g = 9.8$ by default?

In [] :

Classes

- A class contains **attributes** (data) and **methods** (functions) that operate on attributes
 - Classes are defined using the `class` keyword
 - The constructor is a function called `__init__(self, arg1, arg2, ...)` that is automatically called when objects are created
 - `self` represents an object in the class (such as the object being created or accessed)
- In [24] :
- ```
class MyFraction():
 # constructor
 def __init__(self, num, den):
 # class attributes
 self.num = num
 self.den = den
Frac = MyFraction(1, 2) # create a MyFraction object
print(Frac.num) # access the num attribute of F using a dot
1
```
- ### Methods
- Methods are functions that are defined in a class
  - The first argument must be `self`, which is automatically passed when the method is called
  - Methods can be called using a dot
- In [8] :
- ```
class MyFraction():
    # constructor
    def __init__(self, num, den):
        # class attributes
        self.num = num
        self.den = den
    # method to compute the floating-point approximation to the fraction
    def compute_float(self):
        return self.num / self.den
Frac = MyFraction(1, 2) # create a MyFraction object
f = Frac.compute_float() # call the compute_float method; we do not pass it any arguments
print(f)
0.5
```
- ### Class inheritance
- Subclasses inherit the attributes and methods of their parent class (or superclass)
 - Changes to the subclass do not affect the superclass
 - The constructor of the subclass needs to call the constructor of the superclass
- In [9] :
- ```
class NamedFraction(MyFraction):
 def __init__(self, num, den, name):
 super().__init__(num, den) # calling the constructor of the superclass MyFraction
 self.name = name
 def sig_fig(self, n): # add a new method to the subclass
 return round(self.num / self.den, n)
N = NamedFraction(1, 3, 'One third')
print(N.sig_fig(3))
0.333
```

## Exercise

1. Write a class called `Square` which takes as input the length of one side (which is stored as an attribute)
2. Add a method to compute the area of the square.
3. Create a second attribute for the area and have this automatically computed when objects of the class are created.

## Solution

In [ ] :

## File input and output

Use `open`, `read`, `write`, and `close` for reading and writing external files

Mode specifiers:

| Mode | Operation                                                                                               |
|------|---------------------------------------------------------------------------------------------------------|
| r    | Open a file to read. File must exist                                                                    |
| w    | Open a file to write to. If file doesn't exist: create file. If file exists: overwrite contents         |
| a    | Open a file to write to. If file doesn't exist: create file. If file exists: append text to file        |
| r+   | Open a file to read or write to. File must already exist; previous contents will be overwritten         |
| w+   | Open a file to read or write to. If file doesn't exist: create file. If file exists: overwrite contents |
| a+   | Open a file to read or write to. If file doesn't exist: create file. If file exists: append contents    |

## File input and output examples

In [27] :

```
write Hello! in a file called new_file.txt
file = open('new_file.txt', 'w')
file.write('Hello!')
file.close()
```

In [28] :

```
load the contents of planets.txt
file = open('planets.txt', 'r')
for l in file:
 print(l.split())
file.close()
```

```
['Mercury,', '3.7']
['Venus,', '8.0']
['Earth,', '9.8']
['Mars,', '3.7']
['Jupiter,', '25']
['Saturn,', '10']
['Uranus,', '8.9']
['Neptune,', '11']
```

The end!