ComSSA FOP Revision

Week 8 (2023 Edition)

Variables and Data Types

Python has four basic data types:

Integers	`int`	Whole numbers	7, −55, 32000
Floats	`float`	Real numbers	256.0, −35.768
Strings	`str`	Any text or characters	“Rats are cool”
Boolean	`boolean`	True or False	True, False

Variables and Data Types

Python is unusual because:

- You don’t need to declare variables (though may have to initialise them)
- Python can work out which type a variable should be (Dynamic typing)

```
name = "ComSSA"
year = 2022
myFOPGrade = 97.5
ratsRock = True
```

– is a string
– is an int
– is a float, and can be a real number if you work hard
– is a boolean.

Variables and Data Types

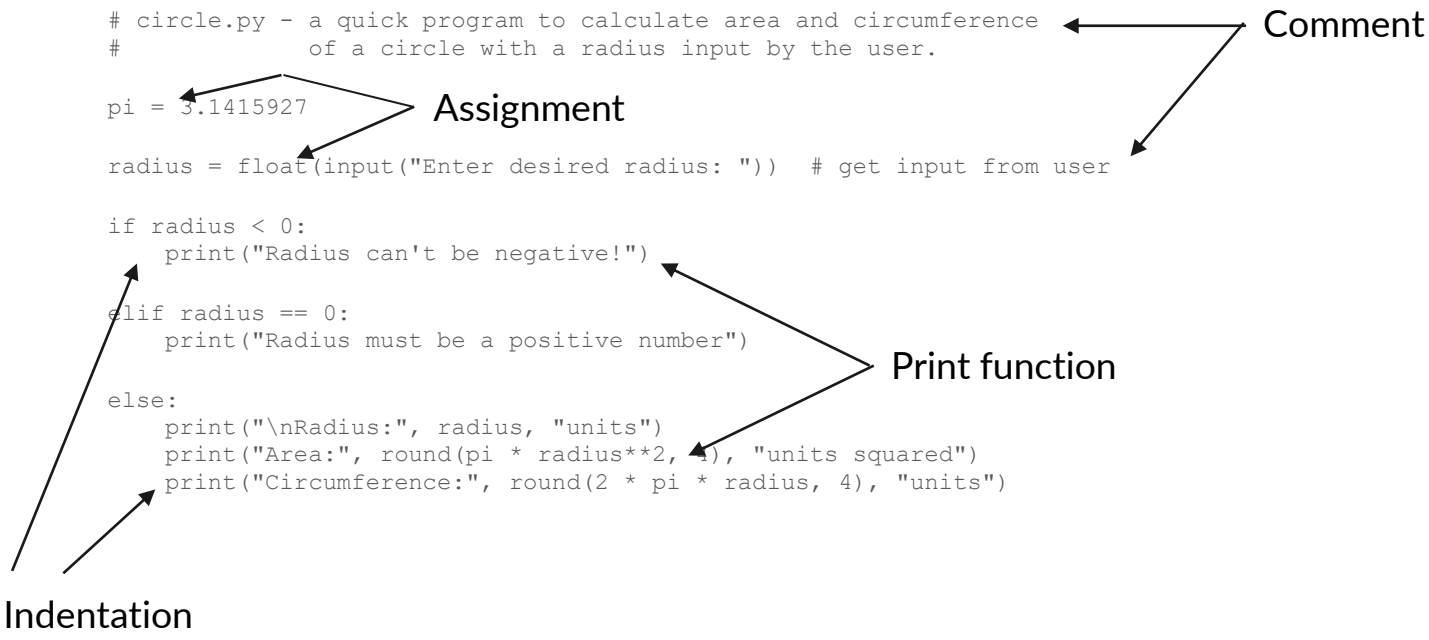
If a variable is an int or float, *it is a number*.

This means you can use it like a number anywhere a number is used!

A Python program

```
# circle.py - a quick program to calculate area and circumference
#             of a circle with a radius input by the user.

pi = 3.1415927

radius = float(input("Enter desired radius: "))  # get input from user

if radius < 0:
    print("Radius can't be negative!")

elif radius == 0:
    print("Radius must be a positive number")

else:
    print("\nRadius:", radius, "units")
    print("Area:", round(pi * radius**2, 4), "units squared")
    print("Circumference:", round(2 * pi * radius, 4), "units")
```

A Python program




```
# circle.py - a quick program to calculate area and circumference
#             of a circle with a radius input by the user.

pi = 3.1415927

radius = float(input("Enter desired radius: "))  # get input from user

if radius < 0:
    print("Radius can't be negative!")

elif radius == 0:
    print("Radius must be a positive number")

else:
    print("\nRadius:", radius, "units")
    print("Area:", round(pi * radius**2, 4), "units squared")
    print("Circumference:", round(2 * pi * radius, 4), "units")
```

A Python program

```
# circle.py - a quick program to calculate area and circumference
#             of a circle with a radius input by the user.

pi = 3.1415927

radius = float(input("Enter desired radius: "))  # get input from user

if radius < 0:
    print("Radius can't be negative!")

elif radius == 0:
    print("Radius must be a positive number")

else:
    print("\nRadius:", radius, "units")
    print("Area:", round(pi * radius**2, 4), "units squared")
    print("Circumference:", round(2 * pi * radius, 4), "units")
```

Type conversion

Boolean expressions

Control structure (If – Elif – Else)

No end!! (This is normal)

More control structures : For loop

A “for” loop looks like this:

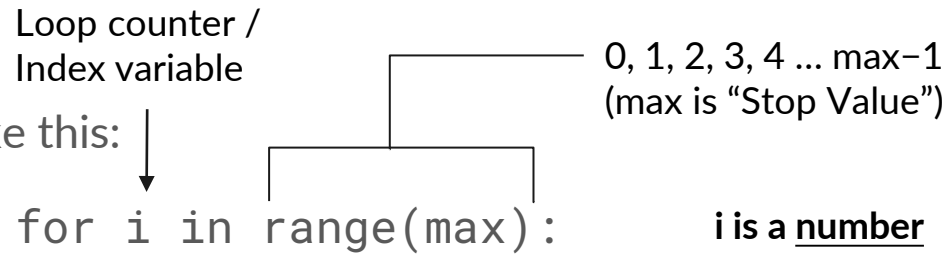
```
for i in range(max):
```

Or can look like this:

```
for word in words:
```

They do different things.

More control structures : For loop



Or can look like this:

```
for word in words:
```

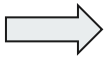
word is part of words

They do different things.

More control structures : For loop

Each time it goes through the loop, it runs the indented code under it.

```
for i in range(5):
    print("Now I am", i)
print("And now I'm done.")
```



```
Now I am 0
Now I am 1
Now I am 2
Now I am 3
Now I am 4
And now I'm done.
```

Or...

```
words = ['this', 'is', 'a', 'loop']
for word in words:
    print(word)
```

```
this
is
a
loop
```

More control structures: While loop

A while loop runs while its expression is True, and stops when it is False.

Have to be careful – Infinite loops (where the condition is always true!)

```
choice = input("Please enter Y or N")
while choice != 'Y' and choice != 'N':
    print("Invalid choice!")
    choice = input("Please enter Y or N")
print("You entered:", choice)
```

!= means "not equal to"

More control structures: While loop

Another example:

```
measurement = 0
total = 0
while measurement >= 0:
    total = total + measurement
    print("Total is:", total, "cm")
    measurement = int(input("Enter a measurement: "))
```

Strings

A string is a series of characters – they could be letters, numbers, or something else entirely. In Python we use double quotes (" ") to note something is a string.

```
myString = "Welcome to the ComSSA FOP Revision Session!"
```

You CAN use a single quote e.g. 'hello', but they’re less flexible.

Never mix them.

Strings and Slicing

If you want to get a single character in a string, you must count from zero. For example, the 2nd character is myString[1].

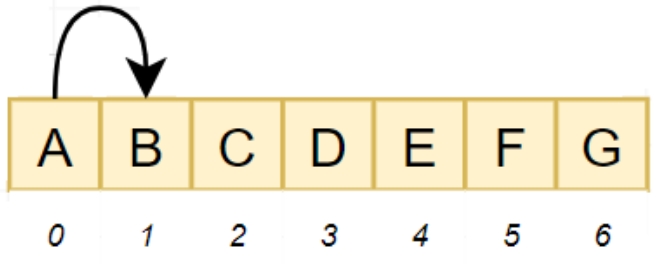
To get substrings, use the following:

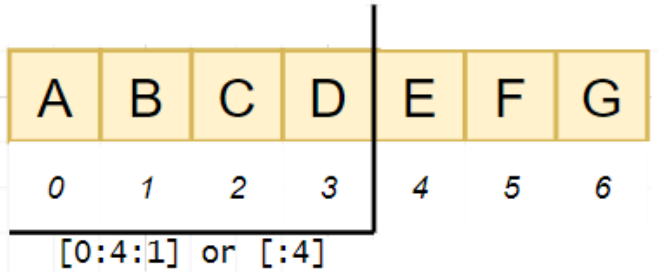
```
start:stop:step
```

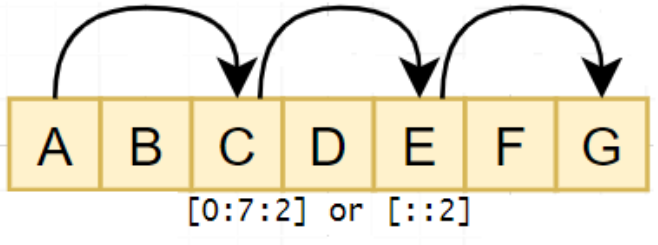
- Start: index of the character it starts on (Default = 0)
- Stop: index of the character AFTER the last one you want to include (Default = end)
- Step: how much you want it to jump each time (Default = 1)

Strings and Slicing

- Start: First character
- Stop: The character AFTER the last one you want to include
- Step: The jump between characters







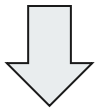
Strings and Slicing

```
myString = "Welcome to the ComSSA FOP Revision Session!"
            0 1 2 3 4 5 6 7 8 9 10...

myString[0]       = ______

myString[5:7]     = ______

len(myString)     = 43     # what does this mean?

myString[42]      = ______

myString[6:-1:-1] = ______
```

Lists

- A structure which can hold any type of item
- Is composed of elements

```
myString = "Welcome to the ComSSA FOP Revision Session!"
myList = myString.split(" ")
```



```
["Welcome", "to", "the", "ComSSA", "FOP", "Revision", "Session!"]
```

Lists

- You can do slicing on lists too:

```
["Welcome", "to", "the", "ComSSA", "FOP", "Revision", "Session!"]

    myList[3]   = _____
    myList[::2] = _____


    myList.append(":D")
    del myList[2]
```

Arrays

Arrays hold an ordered sequence of values.

Unlike lists:

- All of the elements must be the same type
  – for example, all ints, or all strings.
- Once you create it, you can only change the size by creating a new array and copying the elements from one to the other.

Slicing also works on arrays.

Arrays

To use arrays, this must be at or near the very top of your code:

```
import numpy as np
```

You can create arrays by a number of methods:

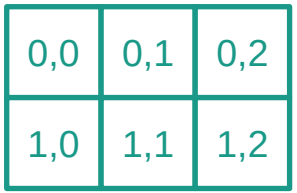
- `myArray = np.array(myList)  # converting from list`
- `myArray = np.zeros(100)     # creates an array`
  `# of 100 numbers,`
  `# all equal to 0`
- `fives = np.arange(0,55,5)   # [0, 5, 10... 50]`

Arrays

You can manipulate arrays:

- `myArray[2:4] = 0` `# sets elements 2 and 3 to 0`
-
```
a = np.array([1, 2, 3])
a3 = a * 1.05
print(a3)                → [1.05, 2.10, 3.15]
```
- `print(a3.max())` → `1.05`
- `print(a3.mean())` → `2.10`

Arrays - Multidimensional



Multidimensional arrays are a lot like normal arrays. The differences:

Declaring:

Double brackets important

```
myArray = np.zeros(100)          my3DArray = np.zeros((100, 100, 100))
myArray = np.array([1, 2, 3])    my2DArray = np.array([[1, 2], [3, 4]])
```

Accessing:

```
myArray[2, :, :]            # means all rows, all columns, 2nd table
myArray[:, :5, :]           # means all rows up to 4, all columns, all tables
```

Arrays - Multidimensional

To loop through multidimensional arrays, we need multiple loops:

```
for row in range(len(myArray[:, 0])):
    for column in range(len(myArray[0, :])):
        print("Value at [", row, column, "] is:”, \
              myArray[row, column])
```

Note `myArray[row, column]` can also be written `myArray[row][column]`

Functions

Functions are a great way of making our code more efficient.

Instead of having to do the same thing multiple times with only slightly different options, we can do it once, then call it multiple times. Not only shorter, but a lot easier to understand.

Functions

A typical function looks like this:

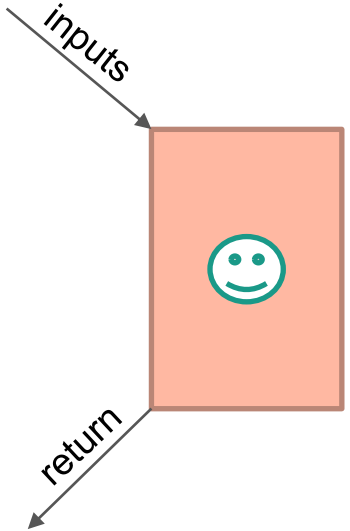
Inputs from outside

```
def sum_of_squares(a, b):
    output = (a * a) + (b * b)
    return output
```

Output sent to outside

It would be called in code like this:

```
myNum = sum_of_squares(3, 5)
```



Functions

Understanding scope is important!

Variables inside a function mean nothing outside of it.

In the last example, if you did “print(output)” in the main code, it would have an error. Likewise, if you tried to do something with mynum *inside* the function, it would not work (or if it somehow did, it may behave unpredictably).

And see how “3” in the main code became “a” in the function,
and “5” became “b”?

Plotting

- Matplotlib works by building up a buffer of things to show, kind of like a storyboard for an animation.

- It then “presses play” when you run `plt.show()`
  (This should usually only run at the very end, as it empties the buffer)

- For a lot of things you should define axes and work on these
  (As we’ve seen in Prac Test 1 and 2).

Plotting

```
import matplotlib.pyplot as plt

plt.title('Numbers')
plt.xlabel('X')
plt.ylabel('Y')

x_list = [1, 2, 3, 4, 5, 6]
y_list = [2, 1, 8, 7, 3, 4]

plt.bar(x_list, y_list,0.9, color='purple')
plt.show()
```

Plotting

What do you think this code does?

```
fig, axs = plt.subplots((2, 2))
axs[0, 0].set_xlim(0, 36)
axs[0, 0].set_ylim(0, 36)
axs[0, 0].set_facecolor("black")
axs[0, 0].set_title("The void.")
axs[1, 1].set_facecolor("blue")
axs[1, 1].plot([0,0], [35,35], "white")

plt.show()
```

Files

Reading and writing to files is essential to programming. Usually, you will be working with CSV (Comma Separated Value) text files, which look something like this:

```
20908070,Joe Bloggs,COMP1005,85.0
20807060,John Smith,NPSC1003,60.0
20607080,Tess Jones,STAT1005,80.0
```

Files

Two ways to open files for reading:

1.
```
with open("file.csv") as f:
    lines = f.readlines()
```

2.
```
f = open("file.csv")
lines = f.readlines()
f.close()
```

Files

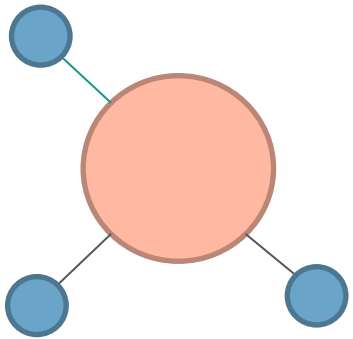
Once we have the data, we need to process it. readlines() creates a list of lines in the file (just long strings, they look meaningful to us but mean nothing to Python). To start processing them, we need to split them up. For example:

```
itemslist = []                          # creates empty list

for line in lines:
    a = line.split(",")                 # split "a,b" into ["a","b"]
    itemslist.append(a[0])              # adds to our list
```

Objects

An object is a instance of a class. Classes have *attributes*.

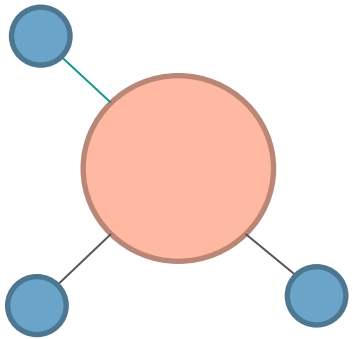
- This room has a building number, a room number, and the number of people it is allowed to hold.
- A cat or dog has a breed, a name and a date of birth.
- A student has a student ID number and a name.



Objects

An object is a instance of a class. Classes also have *methods*.

- Methods are functions that belong to classes. They *do* stuff.
- What are some methods a Cat class might have?
  (Not technical at all – think of an actual cat)



Objects

Do you remember earlier in programming, when you had to put round brackets at the end of some things?

e.g.
```
print( myString.upper() )
myList.append(5)
f.readlines()
plt.show()
```

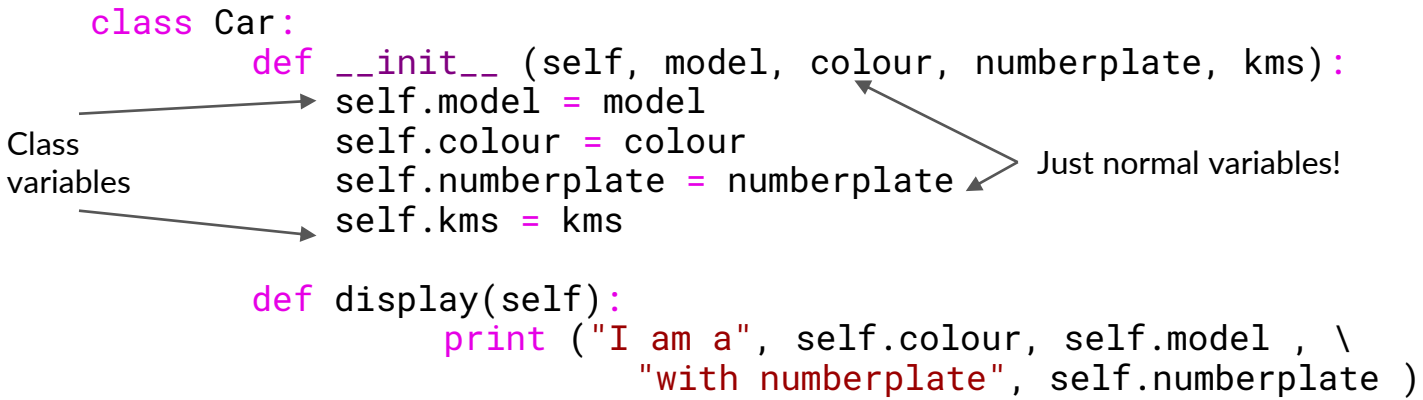
You were using a method!

And if you *didn't* have to use the brackets, you were accessing a variable.

e.g.
```
lights[i,j].colour
```

Objects

Creating a class is easy. We need an __init__ method to initialise the class variables.



Objects

You'll get used to this "self" thing – it appears everywhere. It refers to the particular object created by the class.

```
class Car:
        def __init__ (self, model, colour, numberplate, kms):
            self.model = model
            self.colour = colour
            self.numberplate = numberplate
            self.kms = kms

        def display(self):
                print ("I am a", self.colour, self.model , \
                        "with numberplate", self.numberplate )
```

Objects

All methods have “self” as their first parameter.

So for example:

```
def walk(self, start, end, shoes):
```

We would call this method like this:

```
self.walk(“B314”, “Library”, “hiking boots”)
```

If we didn’t have the “self.”, Python wouldn’t know where to find it or what it belongs to. While it *looks* like a function, it is a *method* of a *class*.

Objects

If we didn’t have initial values, it looks a bit different.

```
class Demo:
    def __init__ (self):
        self.name = ""
        self.start = None
        self.end = None
```

You’ll have to set them later.

Activity 1 – Functions and Arrays

1. Write a function to return the number of characters in your first name.

2. Write a function to return your initials as a list.

3. Write code which creates a 2-dimensional array with 9 numbers in it. Print the middle square.

(Hint for first two – name.split(" ") splits a string by space into a list.)

1	5	6
8	2	7
9	4	3

Solution 1 – Functions and Arrays

1. Write a function to return the number of characters in your first name.

```
def lenFirst(name):
    splitName = name.split(" ")
    return splitName[0]
```

Solution 1 – Functions and Arrays

2. Write a function to return your initials as a list.

```
def initials(name):
    splitName = name.split(" ")
    initials = []
    for i in splitName:
        initials.append(i[0])
    return initials
```

OR

```
    initials = [splitName[0][0], splitName[-1][0]]
```

(no need for `for` loop with this version)

Solution 1 – Functions and Arrays

3. Write code which creates a 2-dimensional array with 9 numbers in it. Print the middle square.

```
import numpy as np

myArray = np.array([[1, 5, 6], [8, 2, 7], [9, 4, 3]])

print(myArray[1, 1])
```

1	5	6
8	2	7
9	4	3

Activity 2 - Objects

Choose a pet – yours or just one you like.

Design a class of that type (e.g. Cat, Dog, Budgie, GuineaPig etc) and give it some reasonable attributes - its name, colour, date of birth for example – there are no right or wrong answers. Write a `def __init__` for it like the ones you've seen in this session.

Now, create an instance (object) for *your* pet using the class you've just written.

Talk with your table group and try some of theirs! Were there any differences with yours?

How would you print or access the variables for your pet? What methods might you write for it? (Note: you don't have to actually write the methods)

Thank you for attending this ComSSA revision session!