INTRODUCTION TO PYTHON

SCHOOL OF COMPUTER SCIENCE, UNIVERSITY OF NOTTINGHAM NINGBO CHINA

LESSON 1: PRIMER

OVERVIEW

In this lesson, we will teach you enough to begin writing your own Python programs. Topics covered:

1. Calculations and variables.
2. Data types (int, str, list, range, bool)
3. If… else …
4. For loops
5. User-defined functions

PRE-REQUISITES

- This lesson assumes you have installed Python3, run a Python session and can save your work.
- If not, then go back to Lesson 1.
- Follow these examples and exercises on your own computer.

CALCULATIONS & VARIABLES

- Calculations can be entered directly into your Python script. Try

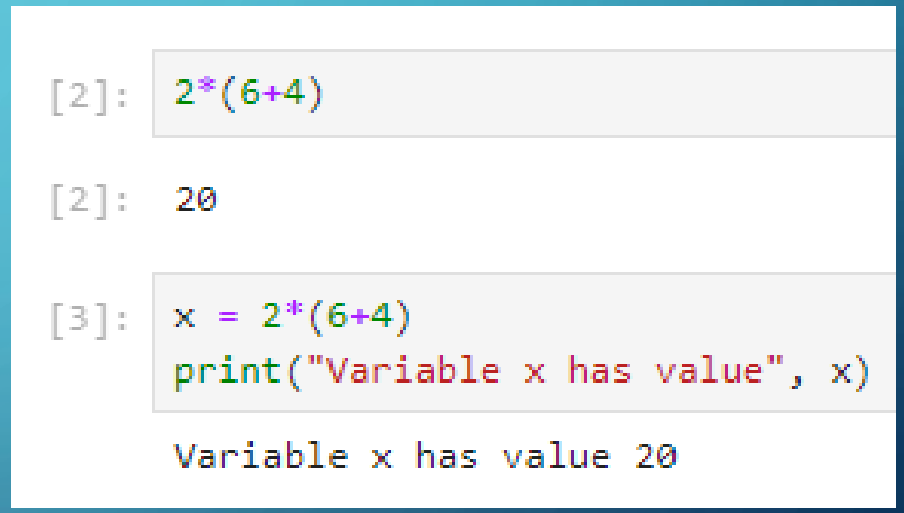
```
2*(6+4)
```

- Variables can be used to store values. Try

```
x = 2*(6+4)
```

- Print values:

```
print("Variable x has value", x)
```



DATA TYPES: STRING

- You can code with different data types. Here's a string:

```
name = "John Cleese"
print("Name is", name)
```

- Use the f-strings to embed values (expressions) inside *formatted* strings:

```
name = "John Cleese"
print(f"Name is {name}")
```

DATA TYPES: LIST

- You can code with lists of values:

```
xs = [4,2,7,2]
print("A list of numbers:", xs)
```

- Extract values at different index values:

```
print("The first number is", xs[0])
print("The third number is", xs[2])
```

- Notice that indexing starts at the value 0 (zero).

DATA TYPES: LIST OF STRINGS

- Lists can take different data types:

```
fruits = ['Apple', 'Banana', 'Pear']
print(fruits)
```

- Get the length of a list:

```
len(fruits)
```

- Notice that indexing starts at the value 0 (zero).

UPDATING A LIST

- Values can be appended to lists:

```
# add a new fruit
fruits.append("Grape")
print(fruits)
```

- Incidentally, notice the hash symbol (#) is used for comments.
- Update the value at a particular index value:

```
# change value in list
fruits[1] = "Plum"
print(fruits)
```

FINDING A VALUE IN A LIST

- Test if a value is in a list:

```
print('Plum' in fruits)
print('Banana' in fruits)
```

- These statements will return:

```
True
False
```

- There is a plum, but no banana.

AGGREGATE FUNCTIONS ON LISTS

- Some in-built aggregate functions exist in Python. This can save you coding time and is computationally efficient.
- For example “sum” can be used to sum the values in a list of numbers. Try

```
print( sum(xs) )
```

- Another function “all” takes a list of Booleans and returns True only if they are all true. Try

```
print( all([True,False,True]) )
```

- Now replace the False in this command with True.

DATA TYPES

- There are many data types. Common ones are:
  - Integer (int),
  - Floating point (float)
  - String (str)
  - List (list)
  - Boolean (bool): i.e. True or False
- Find out the type of a value:

```
print(type(x))
print(type(fruits))
print(type(fruits[1]))
print(type(3==4))
```

USER INPUT

- Read user input:

```
your_name = input("What is your name?")
print("Your name is", your_name)
```

IF… STATEMENT

- Conditional statements can be made:

```
if "Apple" in fruits:
  print("Apple is in the list!")
```

INDENTATION

- Important: the indentation after the “if” is required.
- Everything indented after the “if” is only run if the condition is true.
- Generally, Python is strict with indentation and code needs to line up within a block of code.
- Be consistent using spaces or tabs in your code: do not mix them.

IF… ELSE… STATEMENT

- Conditional statements can be made to run code if the condition is false. Consider this:

```
f1 = input('Enter a fruit: ')
if f1 in fruits:
  print(f1, "is in the list!")
else:
  print("Your fruit is not in the list :(")
print("The end")
```

- Again, indentation is very important and determines blocks of code that will run after the “if” or “else”.

EXERCISE 1

Write code that allows the user to enter the name of a fruit and appends it to the list if it is not already there. Otherwise, it prints “Already on list.”

Example output:

```
Which fruit? Melon
['Apple', 'Banana', 'Pear', 'Melon']
```

COMPOUND CONDITIONS

- Logical operators “and” and “or” can be used to include multiple conditons.
- Logical operator “not” negates a Boolean value.
- Operators “==“, “>” etc. can be used to compare values.
- Consider this:

```
if  xs[2]==5 and not len(fruits) < xs[1]:
  print("There is enough fruit.")
else:
  print("Buy more fruit.")
```

- Try changing the “and” to an ”or”. What happens?

FOR… LOOP

- Loops over ranges of values can be written using a For… loop.

```
for i in range(0,len(xs)):
    print("Element", i, "is", xs[i])
```

- In this code, the variable i iterates over values 0, 1, 2, … to the length of the list -1.
- In each iteration, the next element of xs is printed.
- Again, indentation is important to determine what code is included in the for… loop.

FOR… LOOP

- In this version, the For… loop works across values in a list:

```
for f2 in fruits:
  print(f2)
  if  f2=="Apple":
    print("--> This is the one!!!")
```

- Notice that the if statement is nested inside the for loop.

INDENTATION AND CODE BLOCKS

- Consider this code (based on lists **xs** and **fruits** defined earlier):

```
xs_sum = 0
for i in range(0,len(xs)):
  if  xs[i]>2:
    print("Main fruit is", fruits[i])
    xs_sum += xs[i]
print("The sum is", xs_sum)
```

- Can you describe what this code does?
- Consider the code in bold: what happens if you change the indentation?

EXERCISE 2

- Suppose the list fruits represents fruits in a fruit basket.
- Suppose xs expresses the quantity of each fruit.
- Write code that calculates the total value of a fruit basket if each piece of fruit has the following value: $2 for a banana, $0.20 for a grape, $1.50 for a melon, and $1 for anything else.
- Your code needs to work for any list of fruits.
- Only use the commands and operations you've been taught in this lesson!

USER-DEFINED FUNCTIONS

- Write your own functions like this:

Parameter

```
def triangle_number(x):
    y=0
    for i in range(x+1):
        y += i
    return y
```

- Again indentation is important to determine the block of code to be run in the function.
- The “return” command returns a value from the function.
- Question: why is **x+1** required (not just **x**)?

USER-DEFINED FUNCTIONS

- This example takes a value x, computes a triangle number using a for… loop and then returns the computed value.
- Call the function like this:


```
triangle_number(4)
```

returns `10`.

- Also pass computations as input arguments:

```
triangle_number(int(input("Enter a number:")))
```

USER-DEFINED FUNCTIONS

- User-defined functions can be used in your program.
- For example, this code prints the first triangle number greater than 100:

```
i = 0
t = 0
while t<100:
  i+=1
  t = triangle_number(i)
print(t)
```

- Notice the use of a “while” loop here.

USER-DEFINED FUNCTIONS

- User-defined functions can take more than one argument. Consider:
- For example, this code prints the first triangle number greater than 100:

```
import math
def pythagoras(x,y):
    return math.sqrt(pow(x,2)+pow(y,2))

pythagoras(3,4)
```

- Notice the use of the `math` module. This is required to compute the square root "`sqrt`". The math module needs to be imported in the first line.
- Python programming requires the use of many such modules.

PASS BY VALUE OR REFERENCE

- If the object passed as an argument is immutable (meaning “unchangeable”) then it is passed by value.
- If it is mutable then it is by reference, which means the value can be changed.
- Consider these two examples.

```
def inc(x):
    x = x+1
    return x
y = 2
inc(y)
print(y)
```

```
def apple(L1):
    L1[1] = -1
    return(sum(L1))
z = [1,2,3]
print(z)
print(apple(z))
print(z)
```

- Notice that y is not updated by the function, but z is.

EXERCISE 3

- Following Exercise 2, write a function that will return the price of a fruit when the fruit's name is given as an argument.
- Rewrite the code in Exercise 2 using this price function instead.

FOUND OUT MORE

- This Primer only shows the basics to get started.
- Python is a rich language and there are many more basic features and libraries.
- HOMEWORK: Find out more from web resources and Python programming textbooks.