MANCHESTER 1824

The University of Manchester

PHYS20161: Introduction to Programming for Physicists

Week 2

TurningPoint

Session ID: PHYS20161

Browser: responseware.eu

Can you access this?

Yes, via browser

Yes, via app

Recap

- Lots of information on BB on structure of this course.
- There are BB tests that are part of this course's assessment
  - Do the practice versions first!
- Computers interpret, integers, decimals and strings differently
- We need to check the computer has interpreted the type correctly to avoid errors
- Comments allow another user to understand the code
- Arrays store multiple elements of these types.
  - We start counting at 0

Contents

- Functions
- Scope
- Boolean data types
- if-elif-else loops
- Objects, attributes & methods

**All of this information is available on BlackBoard.*

1st Assignment: Bouncy ball - 10% of course

Your first assignment is to write a code that can calculate the following:

If I drop a bouncy ball from some height, h, and assume it will return to a height ηh after 1 bounce, where 0<η<1; how many bounces will it make over a minimum height, h_{min}?

How long will it take to complete all of these bounces?

The code should ask the user for *h*, h_{min}, and *η*.

You must submit your code on BB by 30/10/20 @ 23:59

This must be your own work. See notes on BB about plagiarism & collusion.

**All of this information is available on BlackBoard.*

1st Assignment cont.

We don’t expect you to be able to write everything for this now.

We estimate this should not take longer than 6 hours to complete.

But you should be able to write parts each week.

Don’t leave it until the last week to start.

We will mark you based on:

- Output (37.5%): whether the code calculates the quantities correctly
- Style (37.5%): how easy it is to follow the code (more next week)
- Extra (25%): whether you have made use of additional features (as shown in lectures) to make your code efficient and succinct.

Functions

A function lets you define an operation you might want to use repeatedly.
They allow you to:

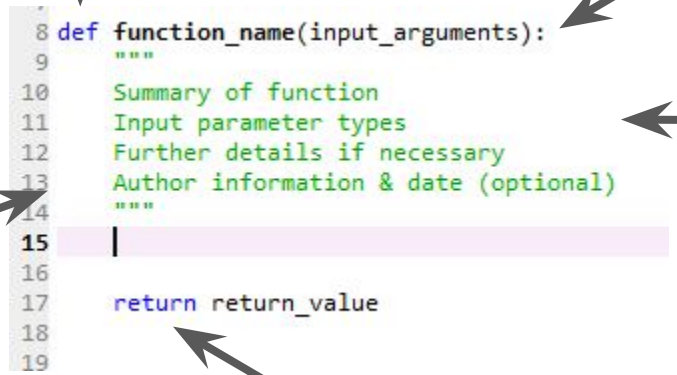
- Separate parts of your code
- Reuse parts of your code
- Organise your code

They are key to good style and allow you to code efficiently and productively.

Functions

Defines the following function with a set of input arguments

Syntax. Won't work without it.



Comments

Everything indented is contained within the function.
4 spaces

All functions MUST return something.
Specify what to return with return

Examples

```
10 def hello_user(user):
11
12     """
13     Greets user
14     user (string)
15
16     Lloyd Cawthorne 29/08/19
17     """
18     greeting = 'Hello ' + user
19
20     return greeting
```

Bug quiz

```
11 def square_number(number):
12     """
13     Squares input
14     number (float)
15
16     Lloyd Cawthorne 29/08/19
17     """
18     number_squared = number**2
19
20 return number_squared
```

Bug quiz

```
11 def repeat_string(string number):
12     """
13     Repeats string number times with space
14     string (string)
15     number (int)
16
17     Lloyd Cawthorne 26/09/19
18     """
19     string_space = string + ' '
20
21     return print(number * string_space)
```

Scope

Scope refers to the visibility of variables. In other words, which parts of your program can see or use. Normally, every variable has a global scope. Once defined, every part of your program can access a variable. It is very useful to be able to limit a variable's scope to a single function.

Example

```
1  # -*- coding: utf-8 -*-
2  """
3  PHYS20161 week 2
4
5  Example of scope
6
7  Lloyd Cawthorne 29/08/19
8  """
9
10
11 def add_ten(number):
12     """
13     Adds 10 to number
14     number (float)
15
16     Lloyd Cawthorne 29/08/19
17     """
18
19     ten = 10.
20
21     result = number + ten
22
23     return result
24
```

Local scope of add_ten.

Variables ‘ten’ and ‘result’ do not *exist* outside of this function.

Example

```
1  # -*- coding: utf-8 -*-
2  """
3  PHYS20161 week 2
4
5  Example of scope
6
7  Lloyd Cawthorne 29/08/19
8  """
9  TEN = 10
10
11
12 def add_ten(number):
13
14     """
15     Adds 10 to number
16     number (float)
17
18     Lloyd Cawthorne 29/08/19
19     """
20     result = number + TEN
21
22     return result
23
```

<u>Using global variables like this is bad practice.</u>

Global scope of programme.

All variables declared here can be used throughout. Provided they are declared **before** use.

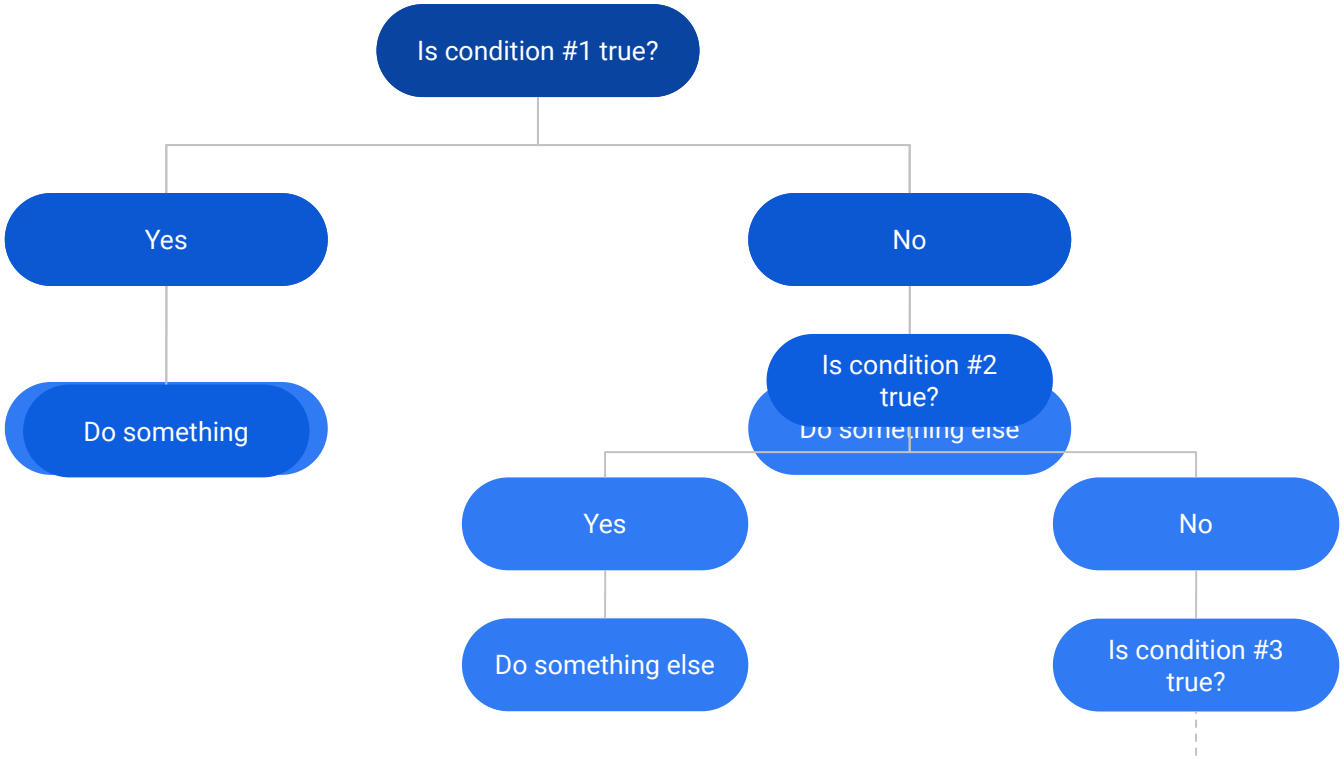
Variable ‘TEN’ exists globally and so can be used by ‘add_ten’.

Style note:
We use UPPER_CASE for global variables, these *should* be constant.

We use snake_case for variables within functions.

Logic operations in programming

Often we want to perform different tasks based on if a condition is true or false.



Boolean data type

A binary variable type:

- True or false
- 1 or 0

Represented as bool in Python with values True or False.

Comparison operators

- == equal to
- != not equal to
- > greater than
- < less than
- >= greater than or equal to
- <= less than or equal to

= vs ==

- = Operator that assigns value to variable.
  - E.g. x = 4
  - x is set to equal 4
- == Operator that compares two values then outputs a boolean (true/false)
  - E.g. x == 4
  - Is x equal to 4?

Logic operators

Sometimes we want to compare multiple aspects.

We can do this using and, or and not.

P	Q	P and Q
True	True	True
False	True	False
True	False	False
False	False	False

P	Q	P or Q
True	True	True
False	True	True
True	False	True
False	False	False

P	not P
True	False
False	True

Logic example, am I working from home?

There is a penguin on my left and today is not Saturday or Sunday

(penguin_on_left) and today != (Saturday or Sunday)

Tomorrow is Tuesday or Friday and it is not Week 1.

(tomorrow == (‘Tuesday’ or ‘Friday’)) and week != 1

Logic question

```
23 def logic_function(bool_1, bool_2):
24     """
25     Outputs bool based on input
26 
27     bool_1 (bool)
28     bool_2 (bool)
29     """
30 
31     return (bool_1 or bool_2) and not (bool_1 and bool_2)
32 
```

Logic question

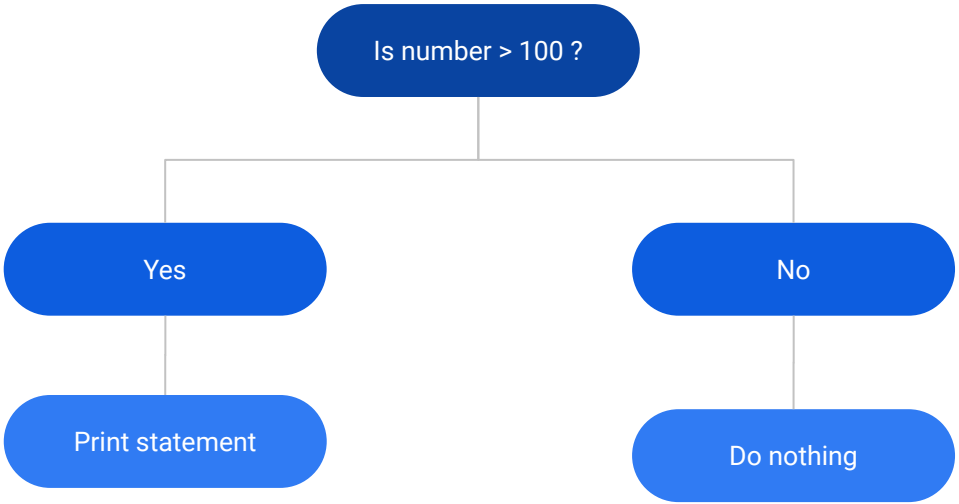
```
23 def logic_function(bool_1, bool_2):
24     """
25     Outputs bool based on input
26 
27     bool_1 (bool)
28     bool_2 (bool)
29     """
30 
31     return (bool_1 or bool_2) and not (bool_1 and bool_2)
32 
```

Logic question

```
23 def logic_function(bool_1, bool_2):
24     """
25     Outputs bool based on input
26 
27     bool_1 (bool)
28     bool_2 (bool)
29     """
30 
31     return (bool_1 or bool_2) and not (bool_1 and bool_2)
32 
```

if-elif-else

Logic functions that test conditions.

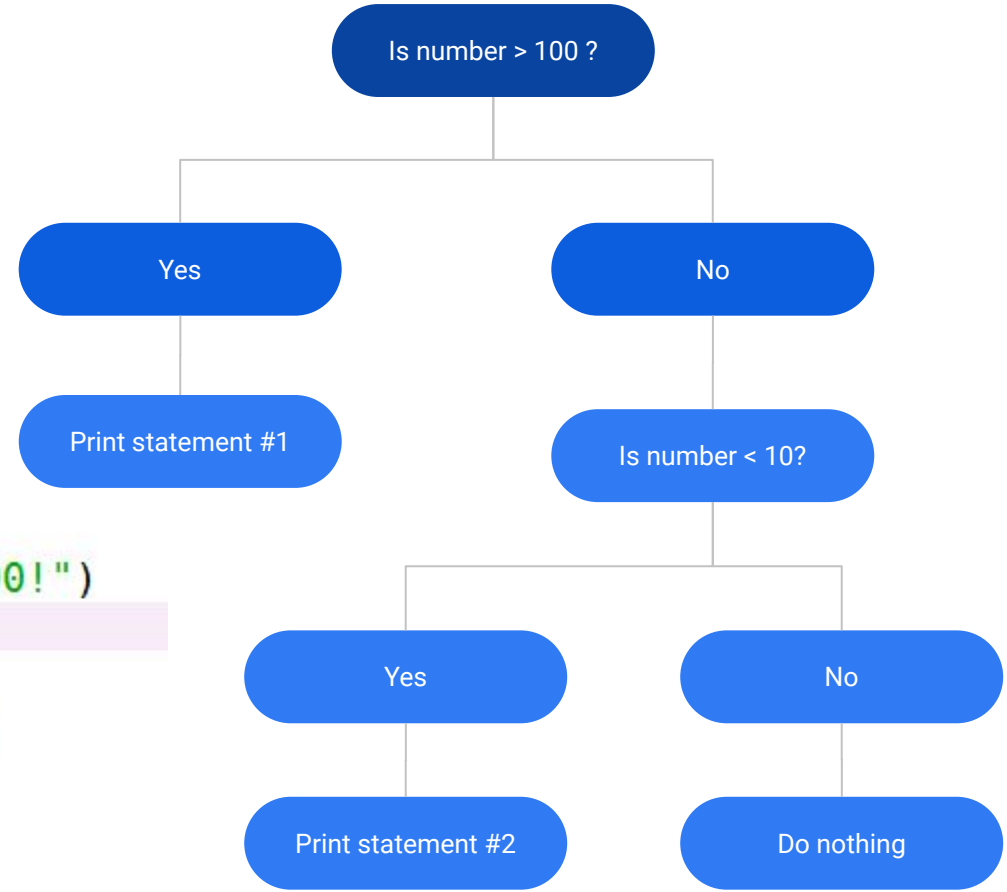



```
10 NUMBER = float(input('Enter a number: '))
11
12 if NUMBER > 100:
13     print("That's greater than 100!")
14
```

if-elif-else

Logic functions that test conditions.

```
19 if NUMBER > 100:
20     print("That's greater than 100!")
21
22 elif NUMBER < 10:
23     print("That's less than 10.")
```

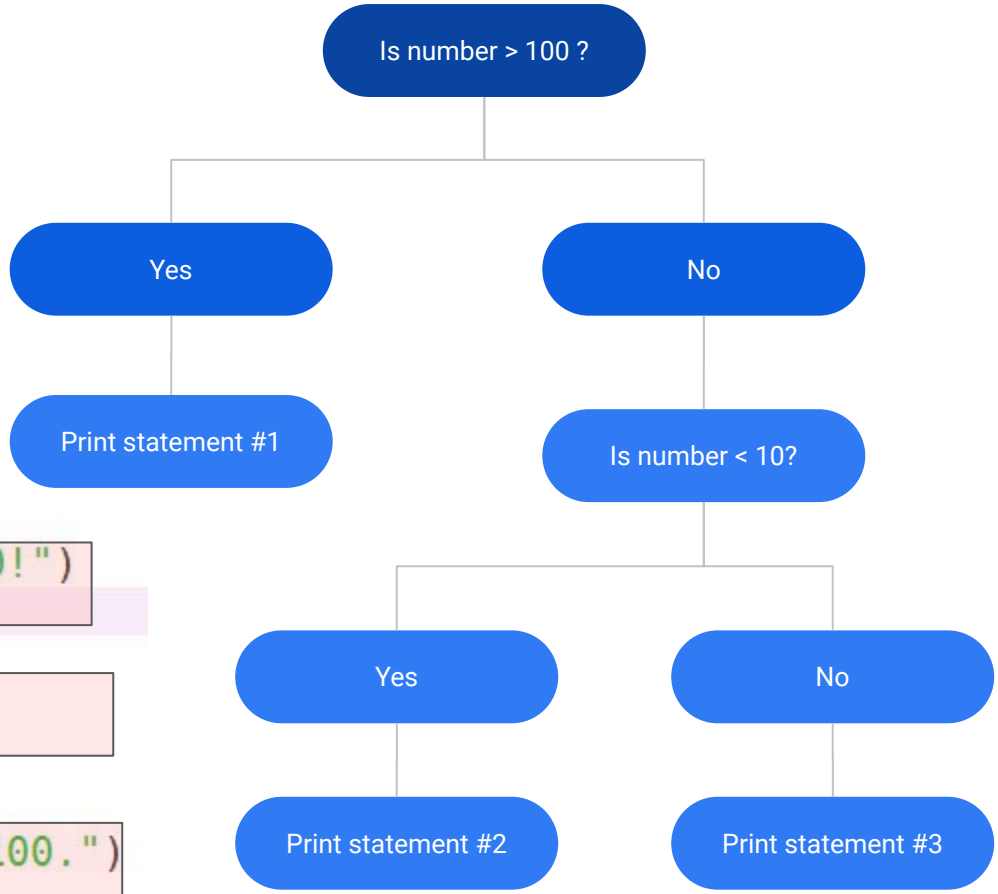


if-elif-else

Logic functions that test conditions.

Local scope of the
if-elif-else statements

```
19 if NUMBER > 100:
20     print("That's greater than 100!")
21
22 elif NUMBER < 10:
23     print("That's less than 10.")
24
25 else:
26     print("That's between 10 and 100.")
27
```

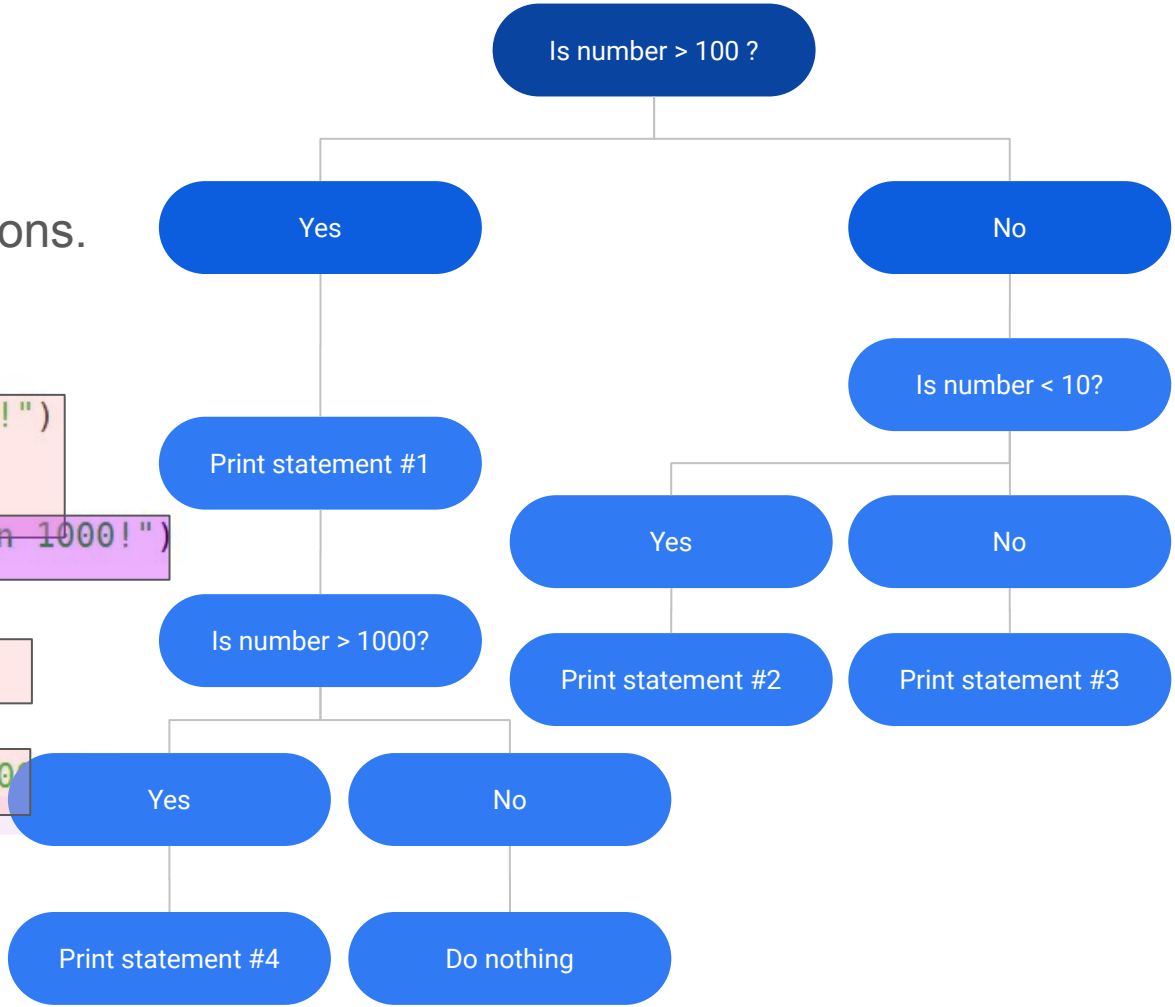


Nested if statement

Logic functions that test conditions.

```
12 if NUMBER > 100:
13     print("That's greater than 100!")
14
15     if NUMBER > 1000:
16         print("... and greater than 1000!")
17
18 elif NUMBER < 10:
19     print("That's less than 10.")
20
21 else:
22     print("That's between 10 and 10
23
```

Nested local scope of
the if statement



Nested if-elif-else statements

Thes can become very messy to the point that they are impossible to decipher.

Try to avoid more than 3 levels of indentation (for any statement). Good use of functions allows this, and makes it easier for a reader (or marker) to understand.

Optional challenge

Can you make a programme that will work for any multiple of 100?

(Only using if-elif-else statements)

Answer next week.

Logic quiz

```
23 def logic_function(bool_1, bool_2):
24
25     if bool_1 == bool_2:
26         return False
27     else:
28         return True
29
```

Example

Examples

Objects

Everything in Python is an object that holds data about itself (attributes) and functions for manipulating the data (methods).

We call these attributes/methods with a full stop, '.'.

```
In [17]: string = 'Hello'

In [18]: string.lower()
Out[18]: 'hello'
```

Your IDE might show you what is available automatically. If not, you can find the list manually using dir(type).

There is an entire course on Object-Oriented Programming (OOP) available in 3rd year.

1st Assignment

You should now be able to ask the user to input the variables and write some functions you will need for the first assignment.

Next week we will cover how to iterate processes with for and while loops.

Quiz this week

Remember to complete the test available on BB.

Attempt the practice version first

Summary

- Functions allow us to define repeated calculations
  - Makes code more readable
  - Makes code more efficient
- The scope of a variable is set by the indentation of where it is declared
  - Can define variables locally or globally to suit our needs
- Can make comparisons between variables to return booleans
- Can use if-elif-else statements to check a series of comparisons
- Everything in Python is an object that has callable attributes and methods