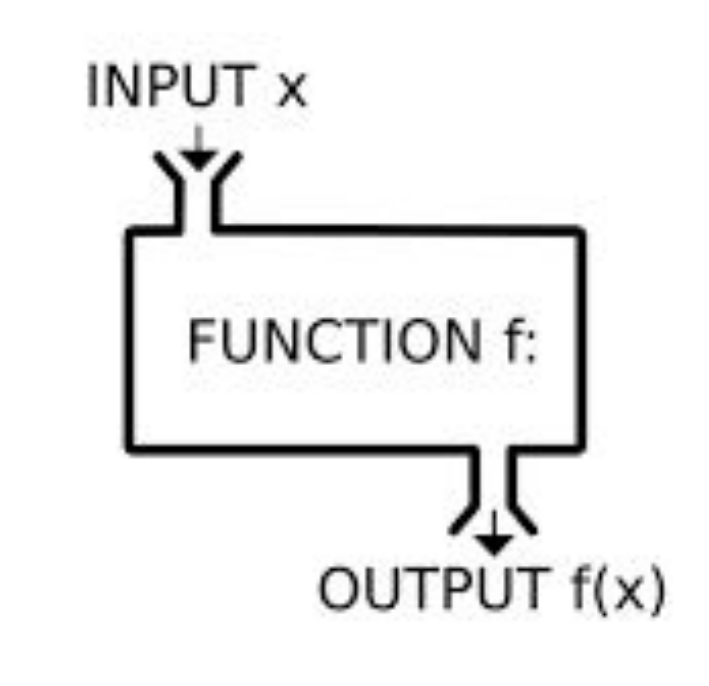
Hello, world!_

MBSI Python Coding Workshop

Lesson 1: Hello World!

Key Terminology



Program

- a set of instructions that a computer can interpret and *execute/run*

Code

- instruction written in a particular programming language

Function

- a reusable block of code that carries out a task. It takes an input(s) (argument), processes it and returns an output

Call

- executing a function within a program
- E.g.

```
function_name(arg1, arg2, arg3)
```

Characters and Strings

Character

- any letter, number, space, punctuation mark, or symbol that can be typed on a computer

String

- Sequence of characters
- Contained by single or double quotes. Keep these consistent!
- E.g.

  ‘coronavirus’ and “coronavirus” are both valid

  “coronavirus’ will produce an error

print() **Function**

Definition:

- Outputs or “prints” a specified message to the screen

Syntax

- print(object(s))

Examples

1. If you would like to print out a single string in a line,

   e.g. Input: `print("Hello World!")` — String

   Output: `Hello World!`

2. Multiple strings in a line,

   e.g. Input: `print("Hello!", "Hello!", "Hello!")`

   Output: `Hello! Hello! Hello!`

Newline Character (\n)

Definition

- A character instructing the Python interpreter to print on a new line

Examples

1. Printing without newline character (\n)

e.g. Input:

```
print("Hello, World!")
print("I'm a medical student from the University of Melbourne.")
```

Output:

```
Hello, World!
I'm a medical student from the University of Melbourne.
```

2. Printing with newline character (\n)

newline character

e.g. Input:

```
print("Hello, World!\nI'm a medical student from the University of Melbourne.")
```

Output:

```
Hello, World!
I'm a medical student from the University of Melbourne.
```

Comments (#)

Definition

- Code annotations that help anyone looking at your code working out what it does and provide further explanation
- *Ignored* by the Python interpreter when running your program

Examples

1. In Python, comments start with the hash (#) character and continue until the end of the line

   e.g.
   ```
   # This is a comment
   ```

2. Can be placed on the same line as an existing statement

   e.g.
   ```
   print('Hello, world!') # This will print out 'Hello, world!'
   ```

3. For multi-line (block) comments you can use consecutive single line comments

   E.g.
   ```
   # This is a block comment.
   # It has multiple lines.
   ```

How many lines of output do we have?

```
print("Hello world!\nI am not a medical student.\n")
```

1	**A**
2	**B**
3	**C**
Error	**D**

Total Results: 0

How many lines of output do we have?

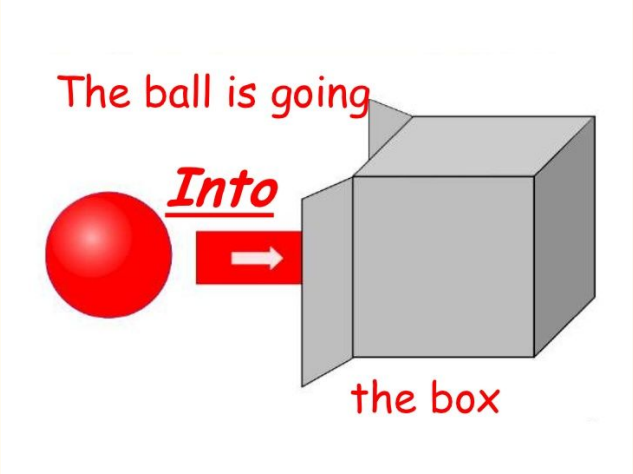
```
print("Hello world!\nI am not a medical student.\n")
```

```
[1]: # Question 1
     print("Hello World!\nI am not a medical student.\n")

     Hello World!
     I am not a medical student.

[2]: print("Hello World!\nI am not a medical student.")

     Hello World!
     I am not a medical student.
```

1	**A**
2	**B**
3	**C**
Error	**D**

Total Results: 0

Variables and Assignment



Variable

- container for storing data values
- A variable is assigned a value using the assignment operator (=)
- Values stored in a variable can be changed

Example

Variables

```
Year_Level = "MD2"
Year_Level = "MD3"
```

Values

assignment operator

Rules for naming variables

Logic

- Have names that make sense and describe the values they are storing

Character

- Only uses the following characters: a-z, A-Z, 0-9, _
- Underscores can be used to separate multiple words

  e.g. `Uni_Melb = 1`

- Can't start with a digit

  e.g. `1VariableName = 1` is invalid

Respond at **PollEv.com/bennyli033**

Text **BENNYLI033** to **37607** once to join, then **A, B, C, or D**

Which naming of the variables is correct?

one variable	**A**
varibable 1	**B**
1 variable	**C**
variable_1	**D**

Total Results: 0

String concatenation

Combining strings together

Example

1. For different strings we can use the *addition operator* (+)

   e.g.
   ```
   print("Hello" + "World!")
   HelloWorld!
   ```

   addition operator

2. To repeat the same string we can use the multiplication operator (*)

   e.g.
   ```
   print("Hello!" * 3)
   Hello!Hello!Hello!
   ```

   number of repeats

   multiplication operator

Printing Multiple Arguments

- Can pass multiple arguments into the print() function
- Arguments are separated by commas (,)
- Arguments can be variables or string constants
- E.g. Input:

```
Name = "Benny"
Age = "21" # in years
print("My name is", Name, ".\nI am", Age, "years old.")
```

variable

comma separating arguments

string

Output:

```
My name is Benny.
I am 21 years old.
```

f-String Formatting

- Begins with f (can also use uppercase F) character followed by the string to be formatted
- E.g. f"string"
- You can include variables in the *f-string* by enclosing them in curly braces {}
- E.g. (Changes)Example outputs

```
print(f"Hello!\nMy name is {Name}.\nI am {Age} years old.")
```

f character

string

variables inside curly braces

input() Function

- Takes user input
- Converts input into a string
- You can provide a *prompt* argument to tell the user what you want them to input
- E.g.

prompt

```
PainScore = input("Rate your pain from 0 (no pain) to 10 (worst pain ever):")
```

prompt

```
Rate your pain from 0 (no pain) to 10 (worst pain ever):10
```

```
print(f"You rated your pain as {PainScore} out of 10.")
You rated your pain as 10 out of 10.
```

variable containing input string

user input

help() Function

- Provides documentation for functions and other objects
- Helps you to find out what a function does and how to use it, including its input arguments.
- E.g.

```
help(print)
```

function you want help with

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

input arguments

purpose of function