Introductory Programming using Python

Day 1

Republic Polytechnic

Welcome and admin matters

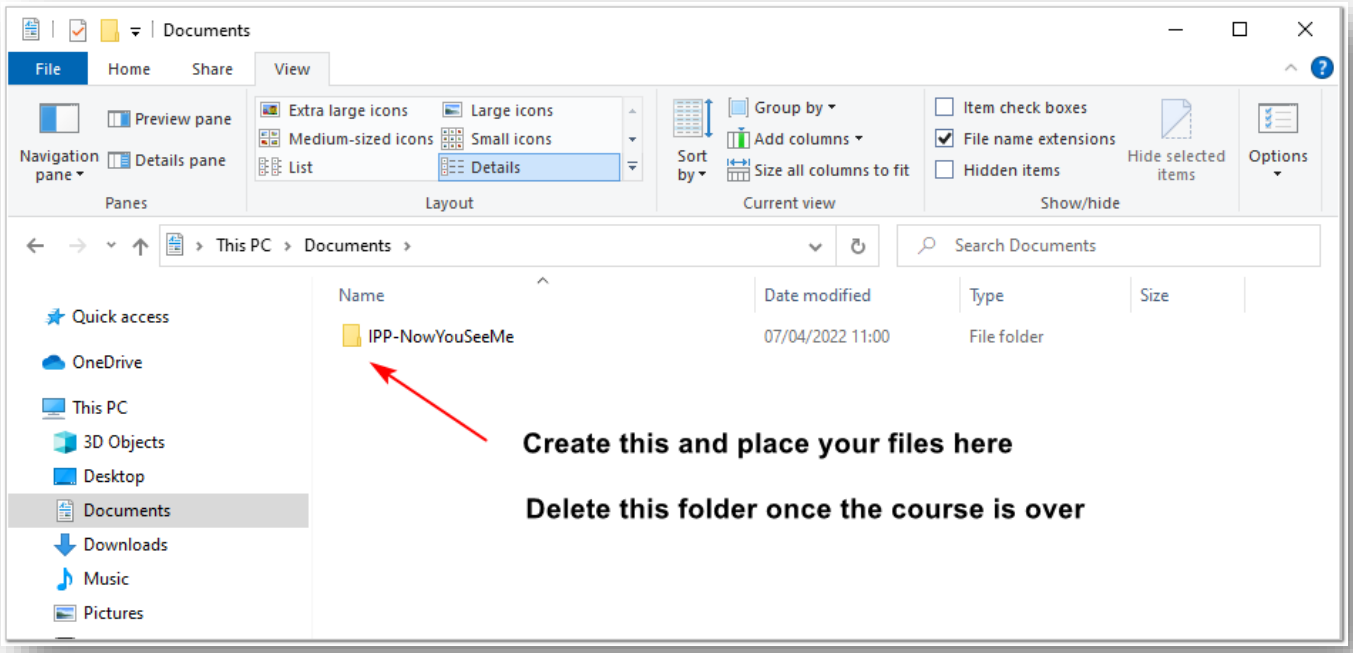
- Please ensure that:
  - your attendance has been captured (via QR code scanning)
  - you have a learning laptop with you
  - you have a good view on the display

- Course material is also available at **https://bit.ly/IPP-MHA**

Img src: https://www.rawpixel.com/image/3283147/free-photo-image-classroom-office-interior

Welcome and admin matters

- A tidy/clean laptop is a good laptop for learning
- Created files on the laptop will linger on. To prevent this:
  - Create a folder in the "Documents" location for this workshop
    - E.g. "IPP-Alan" or "IPP-PeterPan"
    - All user created files to be placed in this folder
  - Delete that folder after the course



Course - Outline

Time	Agenda
9:00 am - 9:15 am	Welcome and admin matters
9:15 am – 10:30 am	
10:30 am – 10:45 am	Break
10:45 am – 12:30 pm	
12:30 pm – 1:30 pm	Lunch
1:30 pm – 3:15 pm	
3:15 pm – 3:30 pm	Break
3:30 pm – 4:45 pm	
4:45 pm – 5:00 pm	Wrap up, Q&A

About This Workshop

- Learn about Python 3, a very versatile and useful language
- Discuss its advantages and disadvantages (also what to look out for)
- Improve your problem-solving skills:
  How to automate the most boring and repetitive stuff using Python
- Have an awareness of available tools and useful modules you can use to build your applications
- It is **NOT** about mastering python programming within 2 days

Pre-reqs and Preparations

For self learning outside of this workshop, please make sure:

- You have a working laptop with Internet access
- You have installed the latest version of Python 3
- You have installed a suitable editor:

  Wing IDE Personal Edition, or other similar tools
- You have Chrome/Edge web browser

Overview: Day One

Morning	Afternoon
• Welcome Message • Variables, Values • Basic Data Types • Data Types Conversion • Display/Outputs • Writing Comments • User Inputs • Decision-Making: if/elif/else	• Lists • Tuples • Repetitions: while vs for • Functions • External Library

Overview: Day Two

Morning	Afternoon
• String functions • String formatting • Dictionary • Read and writing files • Copying, moving and deleting files and folders	• Working with Excel • Connecting to the Web • Demo: Sending Emails (outlook)

What is programming

Definition from Khan Academy:

Programming is **the process of creating a set of instructions that tell a computer how to perform a task**.

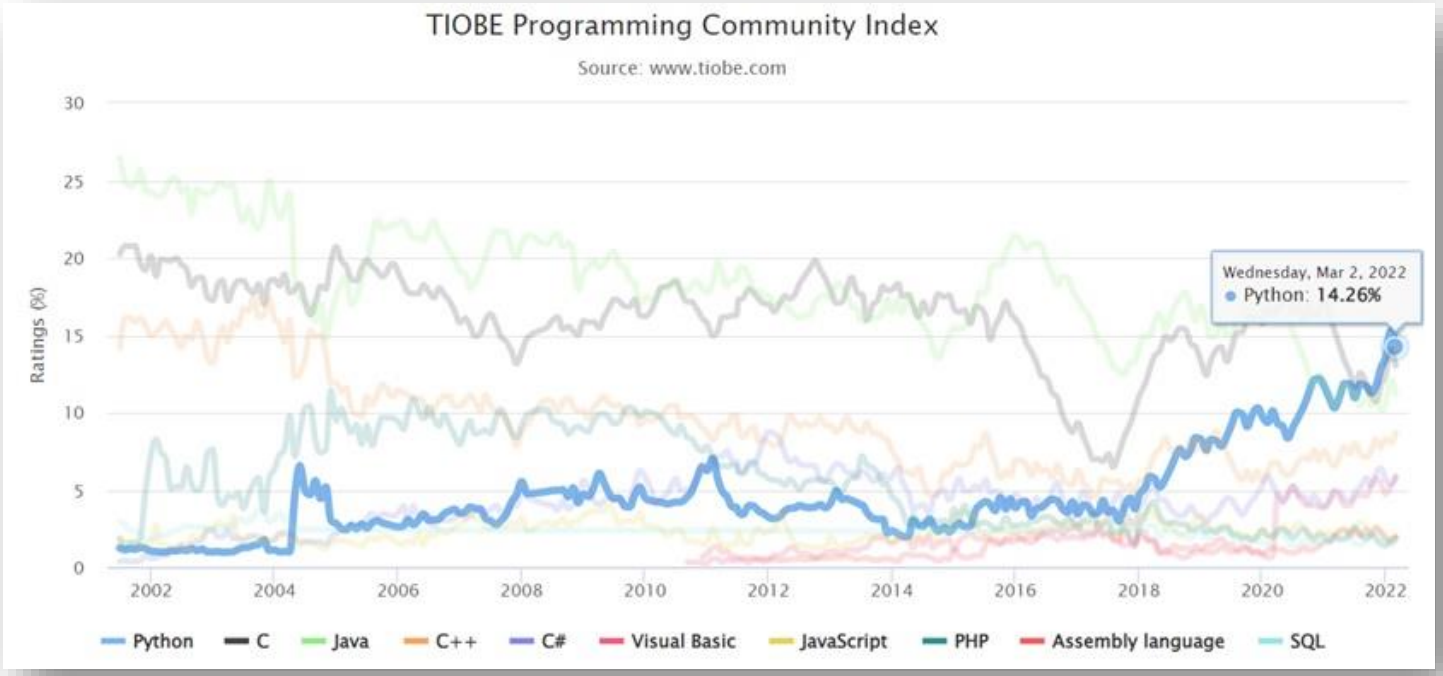

```
1  if marks>10
2    grade = "A"
3
```

Not equal

```
1  if marks>10:
2    grade = "A"
3
```

Introduction

- **What is Python?**
  - Interpreted
  - Interactive
  - Functional
  - Object-oriented
  - Programming language, not just a scripting language

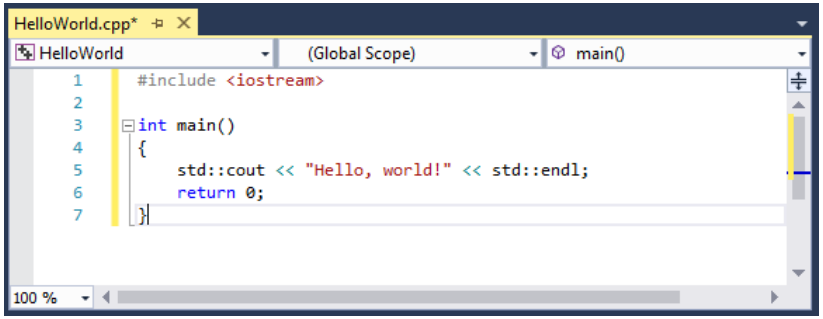


Introduction to Python

- Allows modular programming
- Great emphasis on readability:
  - Code are forced to be **indented**
- Easy to embed in and extend with other languages
- Easy to learn for beginners
- Completely FREE!
- Copyrighted but use is not restricted

```
>>> print ("hello world")
hello world
>>>
```

Vs



Who uses Python?

Ref: http://wiki.python.org/moin/OrganizationsUsingPython

Web Development
- Google (in search spiders)
- Yahoo (in maps application)

Games
- Civilization 4 (game logic & AI)
- Battlefield 2 (score keeping and team balancing)

Graphics
- Industrial Light & Magic (rendering)
- Blender 3D (extension language)

Financial
- ABN AMRO Bank (communicate trade information between systems)

Science
- National Weather Center, US (make maps, create forecasts, etc.)
- NASA (Integrated Planning System)

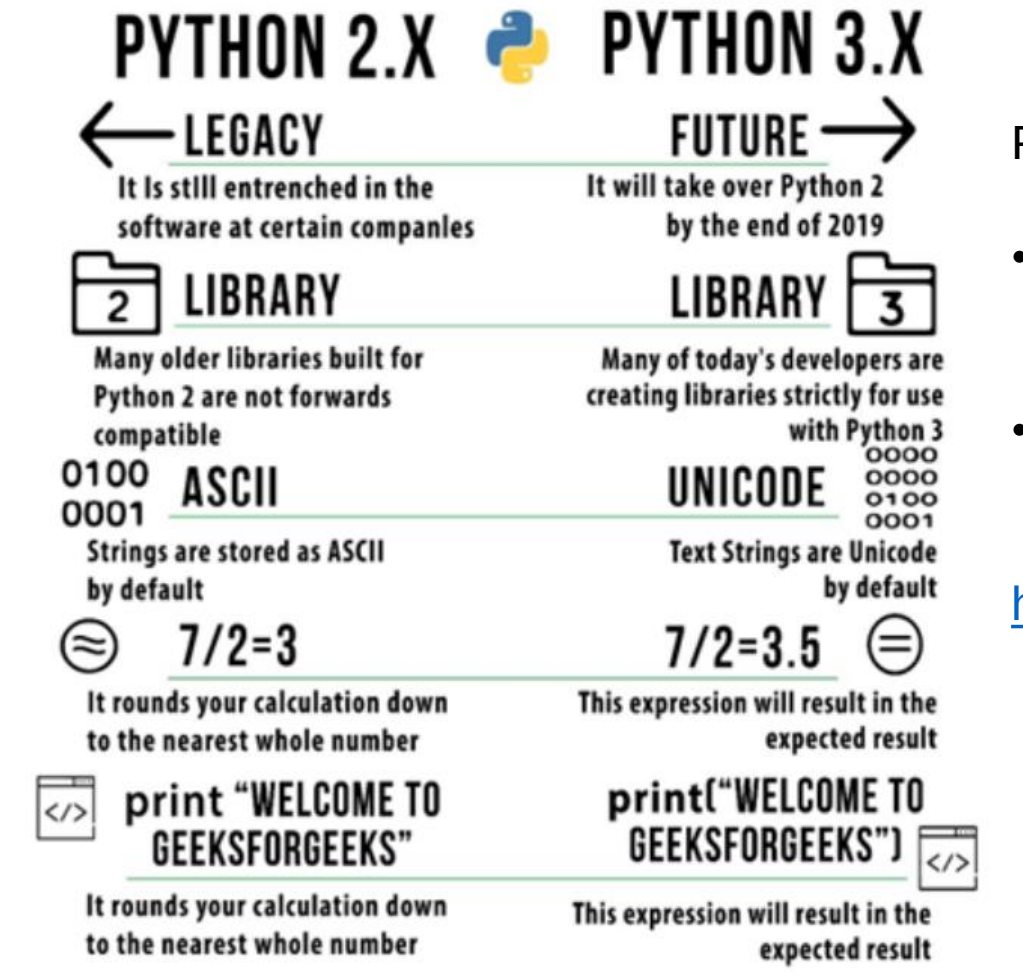
Education
- University of California, Irvine
- University of New South Wales (Australia)
- Republic Polytechnic, Singapore
- National University of Singapore (NUS)
- Singapore University of Technology and Design (SUTD)
- Singapore Management University (SMU)

Why the name, Python?

- Originally not a snake, but from the British comedy "Monty Python's Flying Circus". The snake logo came later.
- Invented in 1990 by Guido Van Rossum
- First public release was in 1991

Python Versions

- Use Python 3.x if you are learning or starting a new project



Python has two major versions

- Python 2 – 2.7.18
  (End of Life – 01 Jan 2020)
- Python 3 – 3.11.1 (current)

https://www.python.org/downloads/

Python 2 vs. Python 3

- **2 distinct versions - Python 2 and Python 3**
  - **Not compatible**

- **Different syntax: e.g. print statement, division**
  - Python 2
    - print "Hello World!"
    - x = 5 / 2 # x's value will be 2
  - Python 3
    - print("Hello World!") # brackets are compulsory now
    - x = 5 / 2 # x's value will be 2.5

- **Which to learn?**
  - Many major frameworks and third-party modules have already migrated or are in the process of moving to Python 3
  - Python 2's EOL is in 2020, no Python 2.8
  - The obvious pick: Python 3

Why Python

- **Focus on problem solving, and not on programming syntax**

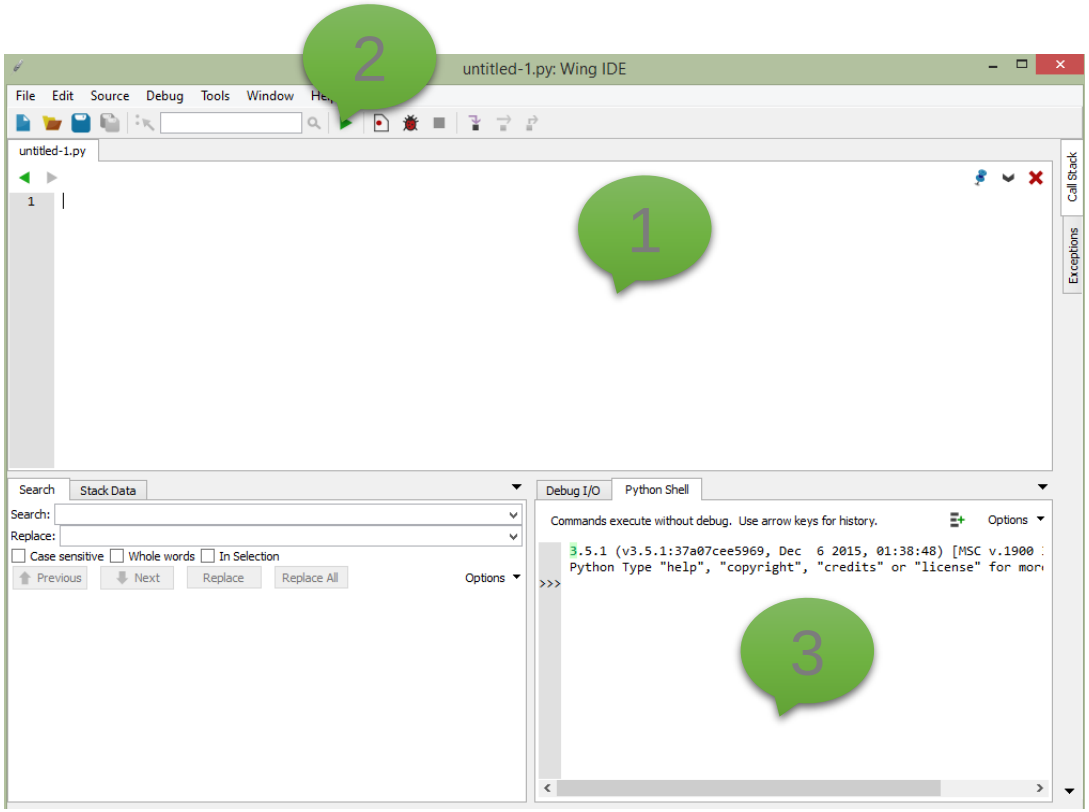
```python
width = input("Enter Width: ")
height = input("Enter Height: ")

area = float(width) * float(height)
print("Area: " + str(area))
```

Python

```java
import java.util.Scanner;

public class AreaApp {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter Width: ");
        double width = scanner.nextDouble();

        System.out.println("Enter Height: ");
        double height = scanner.nextDouble();

        double area = width * height;

        System.out.println("Area: " + area);
    }
}
```

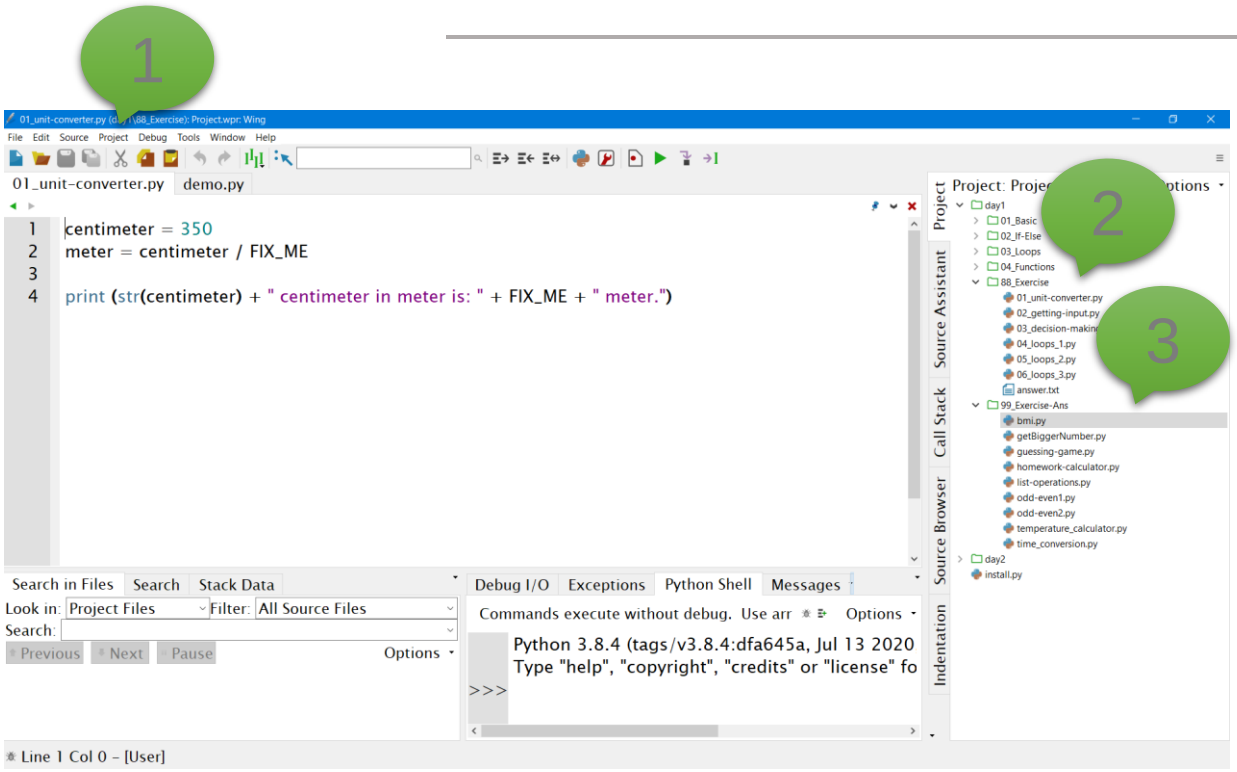
Java

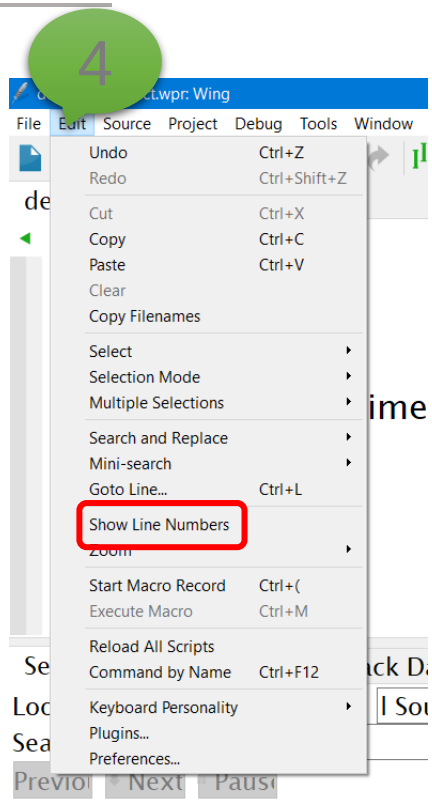
Run Wing IDE



1. Editor
2. Run button
3. Output window / Console

Setting up Wing IDE



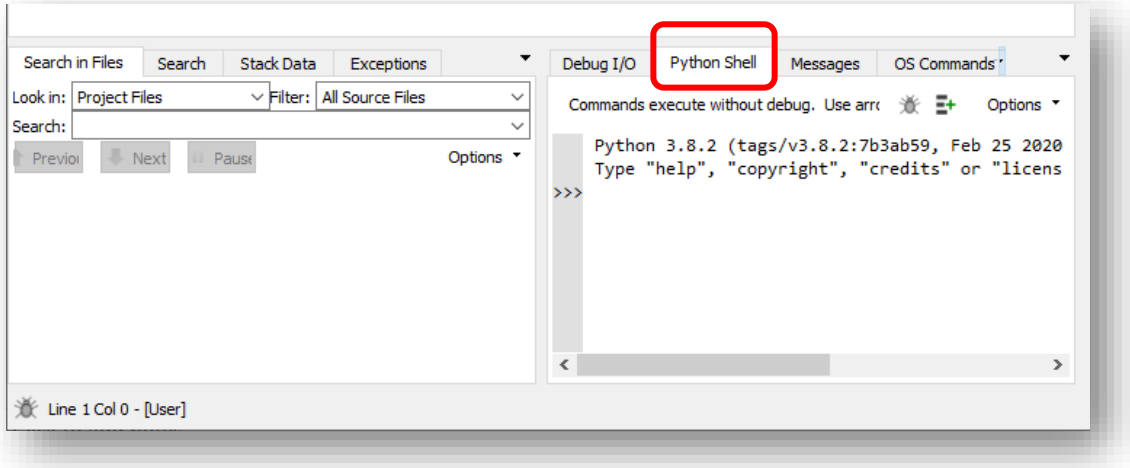


1. Load the project file Project.wpr
2. 8-Exercise contains the skeleton codes
3. Exercise-Ans contains the possible solutions
4. Select “Show Line Numbers” if required

Using the Console

- Also known as the **interpreter**
- See the output straightaway
- Usually used to test very small chunks of code
- Type code after >>>
- To clear the shell, right click in the shell and choose Clear

Let's try!



Exercise 1

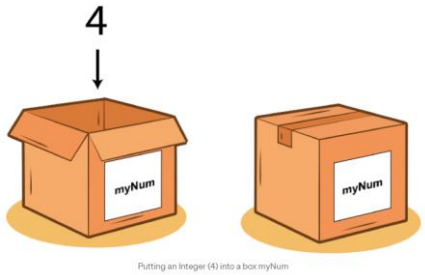
- Perform some simple Mathematics with Python
- Run the following pieces of code in Python interpreter to see how effortlessly Python does it

```
C:\Program Files (x86)\Wing IDE Personal 6.1\bin\dbg\src\debug\tserv
3.8.4 (tags/v3.8.4:dfa645a, Jul 13 2020, 16:30:28) [MSC v.1926 32 bi
Python Type "help", "copyright", "credits" or "license" for more inf
>>> 100 + 10
110
>>> 100 - 10
90
>>> 100 * 10
1000
>>> 100 / 10
10.0
>>>
```

What are Variables?

- Variables are the **storage references** for data
- Some **rules** for naming the variables
  - Case sensitive
  - Cannot start with a number
  - One word
  - Can start with a "_" (underscore)
    - Valid variable names: x, y, abc123, _name
    - Invalid variable names: 1234abc

- To declare a variable to store a piece of data, simply assign a **value** to a name of your choice using the equal (**=**) sign
  - E.g. x = 100



Img src: https://stevenpcurtis.medium.com/what-is-a-variable-3447ac1331b9

Display Variables

- We can use variables after declaring them in our codes
- To display (print) the contents of a variable, use the function **print()**

```
    3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)]
    Python Type "help", "copyright", "credits" or "license" for more information.
>>> x = 100
>>> y = 10
>>> z = x + y
>>> print(z)
    110
```

Data Types

- **Numbers**
  - **int** for whole numbers, e.g. 12, 4, -51
  - **float** for numbers with decimal point, e.g. 5.2, -2.0
- **Text**
  - **str** for a sequence of characters enclosed with either single quote (') or double quotes ("), e.g. "How are you?"
- **Boolean**
  - **bool** **True** or **False** only (without the single/double quotes)
- **Containers**
  - **list** an ordered collection of objects, mutable access using index
  - **tuple** an ordered collection of objects, immutable access using index
  - **dictionary** an unordered collection of objects, access using keys

Basic Data Types

- Examples

int

```
>>> a = 5
>>> b = 2
>>> a + b
7
>>> type(7)
<class 'int'>
>>> a -= 1
>>> a
4
```

float

```
>>> c = 3.0
>>> type(c)
<class 'float'>
>>> 3/2
1.5
>>> 3//2
1
```

str

```
>>> s = "hello"
>>> type(s)
<class 'str'>
>>> s + " world"
'hello world'
>>> len(s)
5
>>> s[0]
'h'
```

Variable and Data Type

- Identify components in a statement

Examples	Variable Name	Data Type	Value
my_name = "alan"	my_name	str	"alan"
age = 25	age	int	25
height = 1.75	height	float	1.75
over_age = True	over_age	bool	True

Conversion between Data Type

- Three important functions: **int**(x), **float**(x) and **str**(x)

Example	int(x)	float(x)	str(x)
x = 1	1	1.0	"1"
x = "alan"	error	error	"alan"
x = 1.5	1	1.5	"1.5"

Mathematics of Programming

- Also known as **operators**
- You can add, subtract, multiply and divide numbers with numbers
  - e.g. 2**+**3, 2**-**6, 2*****3.0, 3**/**2

- Special uses of **+** and *****
  - Add **string** to **string**
  - Multiply **string** with **int**

"hello" + "world"	**→ "helloworld"**
"x" * 5	**→ "xxxxx"**

  - Add **string** to **numbers**?

"5" + 5	**→ Error**

Commonly Made Mistake #1

- It is very common to miss the need to **convert a value to string** during display (print)

Example:

```
age = 25
print( "Age is " + age )
```

String number

Correct approach

```
age = 25
print( "Age is " + str(age) )
```

String String

Basic Arithmetic Operators

Operator name	Code	Example (x = 2, y = 1)
Plus	x + y	x + y will give 3
Minus	x – y	x – y will give 1
Divide	x / y	x / y will give 2.0
Multiply	x * y	x * y will give 2 Use * instead of x for multiplication.
x to power of y	x ** y	x ** y means 2 to power of 1 and the result is 2
Modulus	x % y	x % y will give 0 0 is the **remainder** from 2 divides by 1
Integer division	x // y	x // y will give us the **quotient** when x divides y e.g. 7 // 3 the quotient is 2

Exercise 2

- Identify components in each statement

Statement	Variable Name	Data Type	Value
weight = 65.5	weight	float	65.5
gpa = 3			
gender = "Female"			
enabled = False			
height = 180 + 5.0			
w = float(4) + 3			
x = 7/2			
y = int(4.5) + 5.0			
z = str("1") * 4			
k = "False"			

Comments in Computer Programs

- In computer programming, a comment is a programming language construct used to embed programmer-readable annotations in the source code of a computer program

- **Purpose**:
  - Make the source code easier to understand
  - Document Programmer's intent
  - Explain logic, methods or algorithms

- Ignored by compilers and interpreters
- Syntax: depends on programming language

Python Comments Syntax

- **Inline comment**
  - Symbols or words after the hex symbol # will not be interpreted

- **Block comment**
  - 3 single quotes sequence ''' marks the start/end of a comment block with multiple lines.

```
'''
An example of block comments
The following codes display the
numbers 0 to 9.
'''
numbers = range(10) #An example of inline comments
for i in numbers: #Using a for loop
    print(i)
```

FIX_ME exercise

Warming up

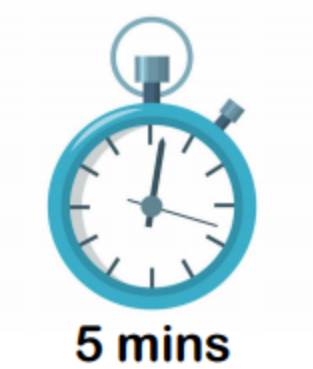
- Convert 350 cm into meter

- Try to fix it, could you?

Exercise – Homework Calculator

- Mick took 3.5 hours to finish his homework. Alice took 2.5 hours to finish her homework.

  Write a program to calculate the total amount of time in seconds that they took to finish their homework.



homework-calculator.py

Exercise – Time Conversion

- Write a program (in 1 script file) to convert 1000 seconds to minutes and seconds.

```
Debug I/O (stdin, stdout, stderr) appears below
Minutes: 16
Remaining Seconds: 40
Time in mins and secs: 16min and 40sec
```

time_conversion.py

Getting User Inputs

- Use **input()** function to ask for user input
- Value entered by the user is stored into a variable as a **string**
- If the value is to be used as a number, you can use the **int()** or **float()** function to convert the value to the appropriate number data type

```
>>> word = input("Enter a word: ")
Enter a word: hello
>>> print(word)
hello
>>> type(word)
<class 'str'>
>>>
```

```
>>> num = input("Enter a Whole number : ")
Enter a Whole number : 8
>>> print(num)
8
>>> type(num)
<class 'str'>
>>> num = int(num)
>>> print(num)
8
>>> type(num)
<class 'int'>
>>>
```

FIX_ME exercise

User input

- A program prompts for a user input, to perform a simple mathematical operation
- The following output will be produced

```
Enter a number : 45
Number received is : 45
One number larger than the number received is : 46
```

- Try to fix it, could you?

Exercise – Temperature Calculator

- The normal human body temperature is 36.9 Degree Celsius.

  Write a program to ask the user for name and temperature and print a message on the screen that indicate the temperature difference from the normal body temperature.

A sample execution of the program

```
Enter patient's name: John
Enter patient's temperature: 37.5
John's temperature is 0.6000000000000014 degree from 36.9 degree celsius
```

Note: We will discuss about the formatting issue of the decimal places in the next slide.

Why 0.30000000000000004?

- 0.1 + 0.2 != 0.3 (surprised?)

- Why? Due to how decimal numbers are stored in computers
  - Floating-point numbers are represented in computer hardware as base 2 (binary) fractions.
  - https://docs.python.org/3/tutorial/floatingpoint.html

  - Conversion between base-2 fraction and floating point numbers
  - https://ryanstutorials.net/binary-tutorial/binary-floating-point.php

- https://0.30000000000000004.com/

Decision-Making

- An **if-else** statement is used to alter the flow of execution of the code

"if" **syntax**:

```
  if  cond : inst
[ elif cond : inst ]
[ else: inst ]
```

```
marks = 30
if marks < 50:
    print("Fail")
else:
    print("Pass")
```

Which code to run?

```
marks = 30
if marks < 50:
    print("Fail")
else:
    print("Pass")
```

- Code between "if" and the colon (:), which is marks < 50 , equates to a True or False value

- If it is of a value True, then the first code, print('Fail') will run
- If it is of a value False, then **else** portion of the code, print("Pass") will execute

- True and False are constants in Python (bool)

Comparison Operators

Expression	What it does
a == b	Evaluates to True when a is equal to b
a != b	Evaluates to True when a is not equal to b
a < b	Evaluates to True when a is lesser than b
a > b	Evaluates to True when a is bigger than b
a <= b	Evaluates to True when a is lesser than or equal to b
a >= b	Evaluates to True when a is greater than or equal to b

```
>>> x = 10
>>> y = 20
>>> print (x == y)
    False
>>> print (x != y)
    True
>>> print (x < y)
    True
>>> print(x <= y)
    True
```

Nested Decision-Making

- A nested if-else statement.
- **"elif**" is a short form for "else if"

```
marks = 30
if marks < 50:
    print("Fail")
elif marks < 80:
    print("Pass")
else:
    print("Excellent!")
```

marks.py

Example of using if/elif/else

- Ask user for the T-shirt size and display the result

```
size = input("Enter your T-shirt Size (s/m/l):")

if size == "s":
    print("You have chosen small size")
elif size == "m":
    print("You have chosen medium size")
else:
    print("You have chosen large size")
```

Notice the implicit assumption made that a user enters only "s", "m", or "l". What if the user enters "xl"?

t-shirt_size.py

FIX_ME exercise

Decision making

- Program 1
  - if a person is 21 and above, this person is an adult
- Program 2
  - Produce greeting based on name
  - Jack -> CEO, Mary -> CFO

- Try to fix it, could you?

Exercise – BMI Calculator

- Develop a BMI Calculator to calculate the BMI of a patient given the weight and height.

$$\text{BMI} = \frac{\text{Weight (kg)}}{\text{Height (m)} \times \text{Height (m)}}$$

Category	Underweight	Ideal	Overweight	Obese
$\text{BMI} = \frac{weight(kg)}{height(m)^2}$	< 18	≥ 18, but < 25	≥ 25, but < 30	≥ 30

15 mins

Summary

Lunch Break

Lists

- Every variable we encountered so far can store only 1 value
- What if we need to store multiple values, e.g. telephone numbers?
- We need to use a variable that is a collection of data, a **list**

- What's unique about Python's lists:
  - Can have multiple data types in the same list
  - Lists are dynamic – can grow and shrink on demand
  - Lists are mutable, i.e. they can be modified after they are created

```
>>> mixed_list = [5, 1.5, "hello"]
>>> mixed_list.append(20)
>>> mixed_list
[5, 1.5, 'hello', 20]
```

Lists

- For example, colours of the rainbow can be grouped under a list data structure.
  - rainbowColours = ["red", "orange", "yellow", "green", "blue", "indigo", "violet"]

- To refer to the individual pieces of data, we can then use
  - print (rainbowColours[1])

- This prints orange, not red! Take note that the index **starts from 0**

Accessing List Elements

```
>>> mylist2 = ["hello", 3.0, 5]
>>> mylist2[0]
'hello'
>>> mylist2[-1]
5
```

- Index starts with 0 and ends with length-1
- Negative indices, starting with -1 are used to refer to elements starting from the last. (-2 for 2nd last, etc.)
- To find out how many elements are there in a list:

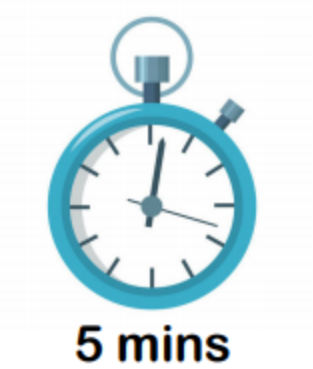
```
>>> mylist3 = ["hello", 3.0, 5, [10, 20]]
>>> len(mylist3)
4
```

List Method Calls

Method	Meaning
`<list>.append(x)`	Add element x to end of list
`<list>.sort()`	Sort the list. A comparison function can be passed as parameter
`<list>.reverse()`	Reverses the list
`<list>.index(x)`	Returns index of first occurrence of x
`<list>.insert(i, x)`	Insert x into list at index i. (same as list[i:i] = [x])
`<list>.count(x)`	Returns the number of occurrences of x in list
`<list>.remove(x)`	Deletes the first occurrence of x in list
`<list>.pop(i)`	Deletes the i^{th} element of the list and returns its value
`x in <list>`	Checks to see if x is in the list (returns a Boolean)

Exercise – List Operations

- Write the code to
  - Create a list with 3 numbers: 1, -5, 15
  - Add the number 20 to the end of the list
  - Remove the number 15 from the list
  - Calculate the total of all the numbers in the list



list-operations.py

Tuples

- Similar to lists, tuples can store multiple values with one variable
- BUT, the items in a tuple are **unchangeable**
  - No adding, removing, updating of the items is allowed

```
gradesTuple = ( "A", "B+", "B", "C+", "C", "D+", "D", "F")
```

- Tuples are declared using parenthesis (), and individual item can be accessed using their index

```
print(gradesTuple [0]) #displays "A"
```

- A tuple can be "unpacked" too

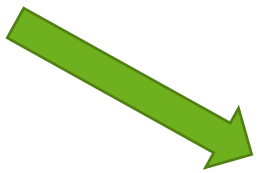
```
( A, Bp, B, Cp, C, Dp, D, F) = gradesTuple
```

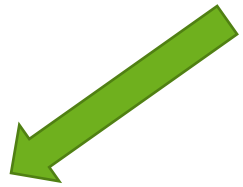
Repetitions – *while* loops

- Instead of writing 5 lines of print() command, you can use a ***while* loop** to execute 1 line of print() command 5 times to generate the same output

```
1  print("Hello World!")
2  print("Hello World!")
3  print("Hello World!")
4  print("Hello World!")
5  print("Hello World!")
```

```
1  i = 0
2  while (i < 5):
3      print("Hello World!")
4      i = i + 1
5
```





```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

while loops - Syntax

- The syntax of the *while* Loop in Python is:

```
while <condition>: # Header ends with a colon (:)
    <statement(s)> # Indented Body
```

- *while* is the Python keyword to represent a while loop
- condition is evaluated to determine if statement(s) are to be repeated. Note the colon at the end of the line
  - When the condition becomes false, the program control passes to the line immediately following the repetition structure or loop
- statement(s) may be a single line or multiple lines of code which are **indented**

while loops – Syntax Explained

Value of "i" is set to 0 before loop starts.
This is the **initialization**.

The **condition** for repeating the while loop
Continue if "i" is less than 5 but
Terminate if "i" is equal or greater than 5

```
i = 0
while i < 5:
    print(i)
    i = i + 1
print ("bye")
```

while loop header that ends with a colon

Indented statements.
In this case, 2 lines of code that are indented make up the block of code to be repeated

Output

```
0
1
2
3
4
```

Increase value of "i" by 1 at each repetition.
This is to ensure that one moves towards **termination** of the repetition. If this line of code is not present, the loop will not terminate.

Statement outside the while loop. This statement is executed when the condition of the loop is false

FIX_ME exercise

While loop

- Print “Hello world” 5 times

- Try to fix it, could you?

Exercise – *while* loop

- Write a program that displays 10 numbers from 1 to 10 using a *while* loop. The number increases by 1 each time.

  The program also calculate and display the sum of these 0 numbers at the end.

Sample of the expected program execution

```
number: 1
number: 2
number: 3
…
number: 10
Total is 55
```

Repetitions – *for* loops

```
>>> numbers = range(10)
>>> for i in numbers:
...     print(i)
...
0
1
2
3
4
5
6
7
8
9
>>>
```

- *for* loops often go hand-in-hand with lists
- Every object in the list will be processed by what is inside the *for* loop
- What is the data type of *i*?

Notice how each call of print at each loop will print at a different line.

How do we print numbers 0 to 9 all on the same line (0123456789)?

for-loop.py

for loops - Syntax

- The syntax of a *for* Loop in Python is :

```
for i in range(start, end, step): # Header ends with a col
    <statement(s)> # Indented Body
```

- *for* is the Python keyword for the for-loop
- The number of repetitions is determined by the **range( )** function (described next). Note the colon at the end of the line.
- statement(s) may be a single line or multiple lines of code which are **indented** (like the while loop)

range function

- Three versions:
  - range(y)
    - starts at 0
    - ends **before** y
    - step up by 1
  - range(x, y)
    - starts **at** x
    - ends **before** y
    - step up by 1
  - range(x, y, s)
    - starts **at** x
    - ends **before** y
    - step **up** by s

```
>>> print(list(range(10)))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> print(list(range(1,10)))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> print(list(range(1,10,2)))
[1, 3, 5, 7, 9]
>>>
>>> print(list(range(10,1,-1)))
[10, 9, 8, 7, 6, 5, 4, 3, 2]
>>>
```

Note: if `s` is negative, then step down by its absolute value

FIX_ME exercise

For loop

- Print “Hello world” 5 times

- Try to fix it, could you?

Repetitions – *for* loops

- A **string** is a sequence, like a list
- *for* loop works similarly with strings

```
>>> s = "freedom"
>>> for c in s:
...     print(c,end=" ")
...
f r e e d o m
>>>
```

for-loop.py

FIX_ME exercise

For loop on String

- Print M, y, o, i from "Mary Poppins"

- Try to fix it, could you?

Exercise – Even/Odd Counter

- **Write and test a program that will read 10 positive integer numbers, determine if it is even or odd, keep count of the number of even and odd numbers and display the final outcome as follows:**

```
Enter number 1: 12
Enter number 2: 7
. . .
Enter number 10: 67
Number of even numbers: 4
Number of odd numbers: 6
```

odd-even1.py odd-even2.py

Sentinel-Controlled Loops

- Required when the number of iterations (repetitions) is not known in advance
- Dependent on user responses
- Only *while* loop can be used
  - An example, to repeatedly ask a user to enter a number. Program stops only after a positive number is entered.

```
num = float(input("Please enter a positive number: "))

while num < 0:
    print("You have entered a negative number.")

    num = float(input("Please enter a positive number: "))

print("The number you have entered is " + str(num))
```

Introduction to Function

- Functions are little self-contained programs that perform a specific task
- You have to **define** a new function before you can use it

Define a function →

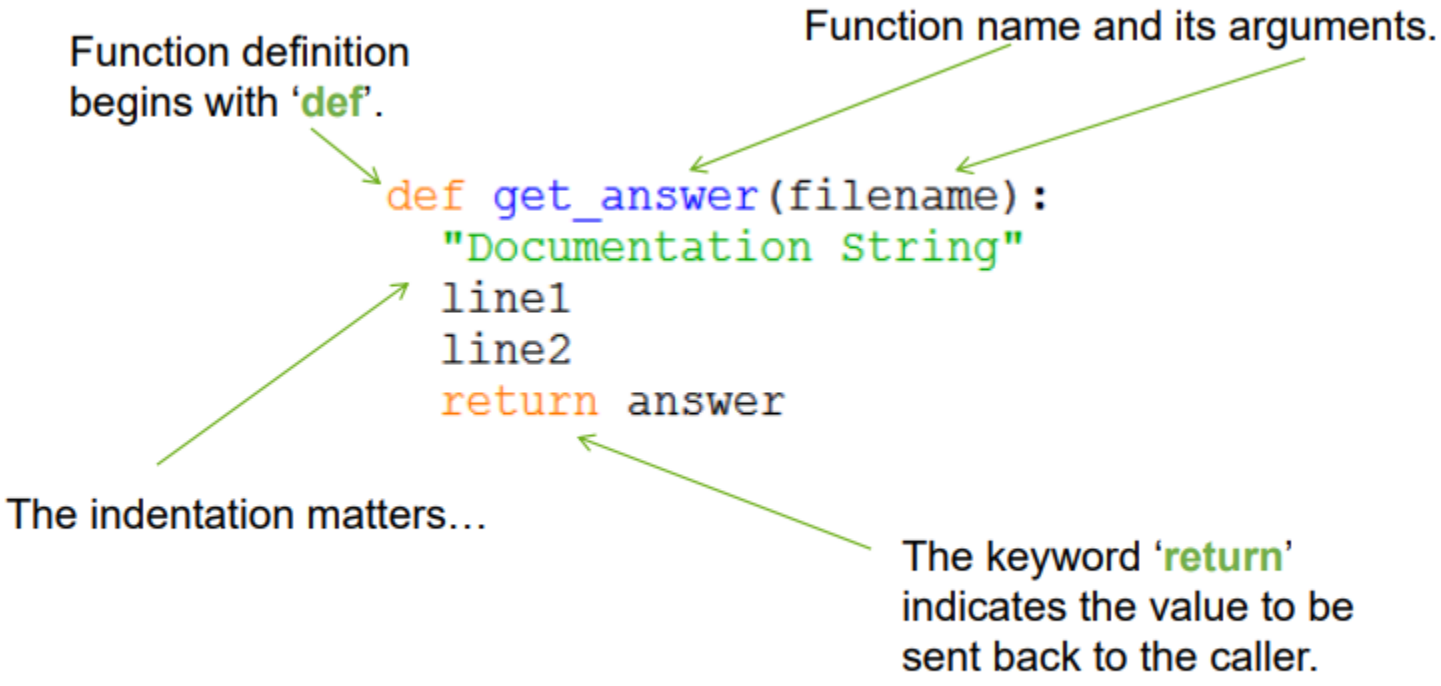
```
def cal_area(width, height):
    return width * height
```

Use a function →

```
area = cal_area(5, 8)
print("The area is " +  str(area))
```

Defining a function – *def*

- No type declarations needed
- Python will figure it out at run-time



No header file or declaration of types of function or arguments.

Why function?

- Function to calculate area of circle based on a given radius

```
def cal_area(radius):
    area = 3.142 * radius * radius
    return area
```

- Uses of function
  - reduce repetitive code
  - define new command by grouping existing commands
  - function name can provide more meaningful name to a series of commands

Example – User defined function

```
>>> def sayHello():
...     print('Hello')
...
>>> sayHello()
Hello
>>> def addNumbers(x, y):
...     return x + y
...
>>> z = addNumbers(3, 4)
>>> z
7
```

Note the differences:
sayHello() does not return a value

addNumbers() return a value

Returning value from Function

- Compare the two functions:
  - **return**: Get back a value after calling a function, assign this value to a variable
  - **print**: Display a value to a user

```
def cal_area(width , height ):
    return  width  * height
```

return a value

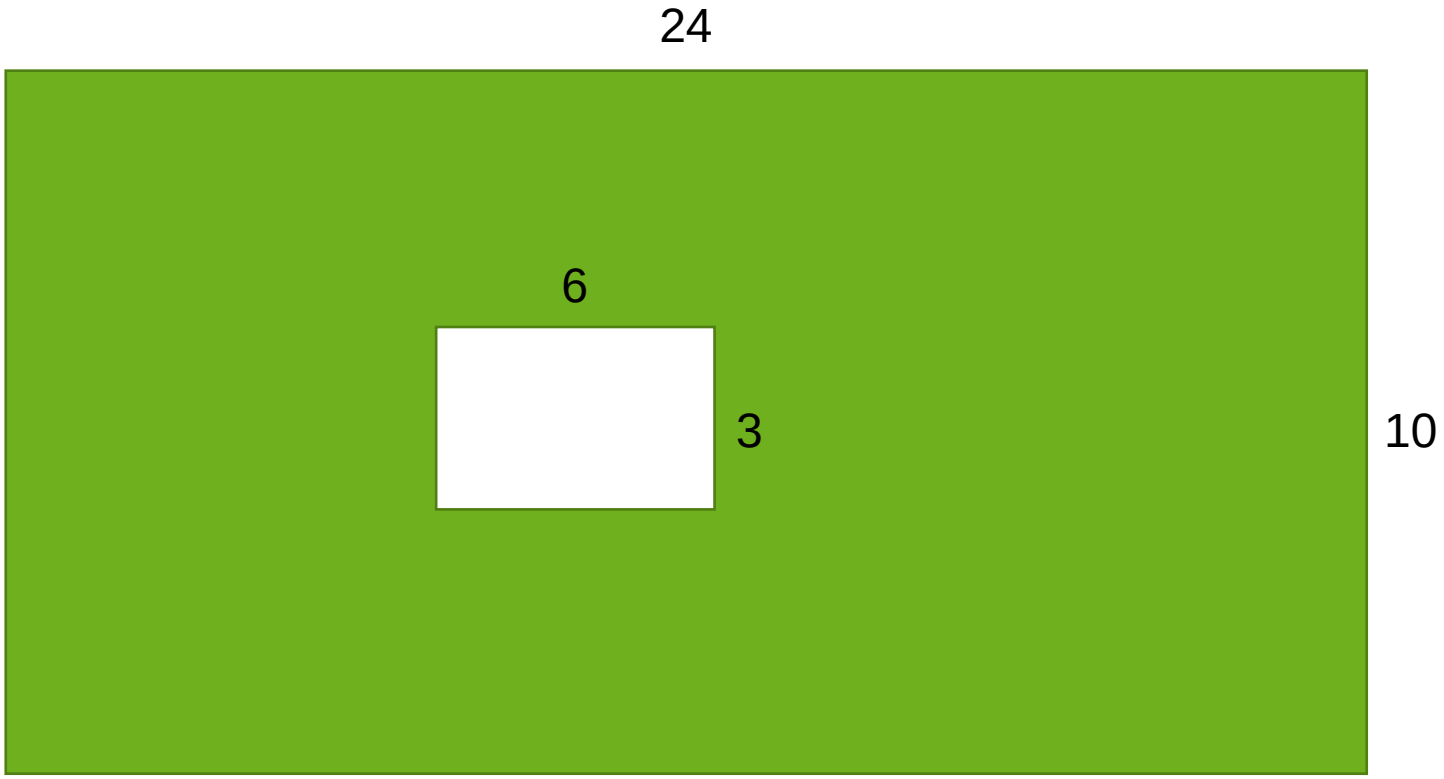
```
def cal_area(width , height ):
    print (width * height )
```

print a value

- Most functions should return instead of print a value
- How a function should work is dependent on the design of the programmer

Example – Calculate area

- Write a program to calculate the area of the shaded region



Example: Function with return value

- Create and use a function to calculate area of rectangle

Define the function

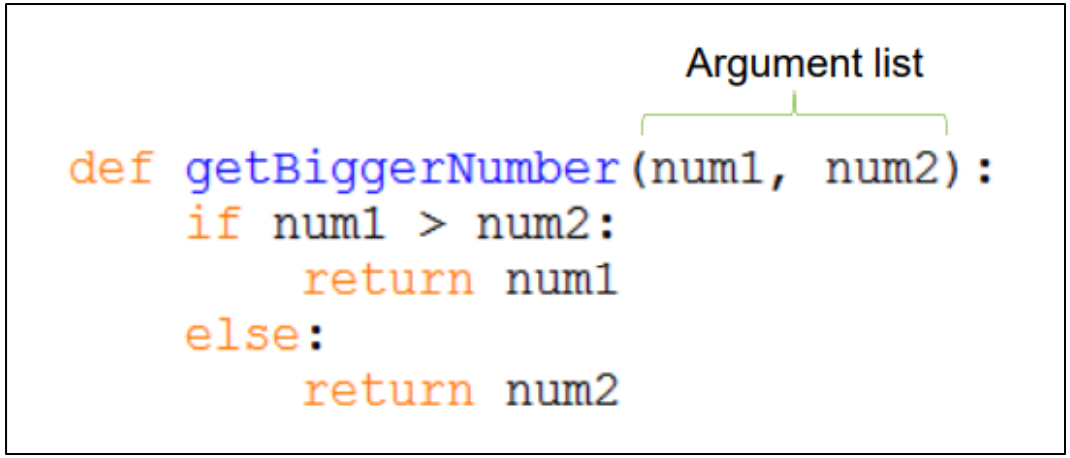
```
def cal_area (width, height):
    return width * height
```

Using the function

```
area1 = cal_area(24, 10)
area2 = cal_area(6, 3)
shared_area = area1 - area2
print("Shared area: " + str(shaded_area))
```

Example: Define and Use Function

- Write a function that takes in two numbers as arguments and returns the bigger number.



getBiggerNumber.py

Exercises on Function

1. Write a function that takes in a number as argument, and returns that number
2. Write a function that takes in a number as argument, and returns that number incremented by 1
3. Write a function that calculates and returns the double of the number given as argument

Optional

More Exercises on Functions

- Write a function to calculate the discounted price given the original price and the discount in percentage.
- For example, if an item costs 100 dollar, and given 10% discount, the function will print a value of 90.0.

Samples:

```
>>> get_discount(100, 10)
90.0
>>> get_discount(50, 20)
40.0
```

get_discount.py

- Write a function that takes in a list of number and return the sum of the numbers.

Samples:

```
>>> get_sum([1, 2, 3, 4])
10
>>> get_sum([3, 3, 3])
9
```

get_sum.py

random library

- A Python library, or sometimes known as package, contains reusable code
- *random* is a built-in library to make random numbers

```
import random

x = random.randint(1, 60)
```

The library **must** be imported first.

Before the relevant functions is used.

random library

- random.randint(a, b)
  - Return a random integer N such that
  - a <= N <= b
  - e.g. *number = random.randint(1, 60)*

- random.random()
  - Return the next random floating point number in the range [0.0, 1.0]

- Other random functions
  - random.shuffle(List) – Re-order all the items in the List randomly
  - random.choice(List) – Returns a random item from the List

More at http://docs.python.org/library/random.html

Exercise – Guessing Game

- Create a random number between 1 and 20 and prompt the user to guess the secret number. He is allowed a maximum of 6 guesses after which the secret number will be displayed and the program exits.

  For every guess, the program will display a message saying if the number guessed is higher or lower than the secret number. If he guessed the correct number, the program will display the number of tries he had taken and the program exits.

  Refer to next slide for samples of the expected program execution.

guessing-game.py

Exercise - Guessing Game (Sample outputs)

Sample 1

```
What is your name?
John
Well, John, I am thinking of a number between 1 and 20
Take a guess
5
Your guess is too low.
Take a guess
10
Your guess is too low.
Take a guess
15
Your guess is too high.
Take a guess
12
Your guess is too low.
Take a guess
14
Good job, John! You guessed my number in 5 guesses!

Process finished with exit code 0
```

Sample 2

```
What is your name?
John
Well, John, I am thinking of a number between 1 and 20
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
nope. The number I was thinking of was 6

Process finished with exit code 0
```

End of Day 1