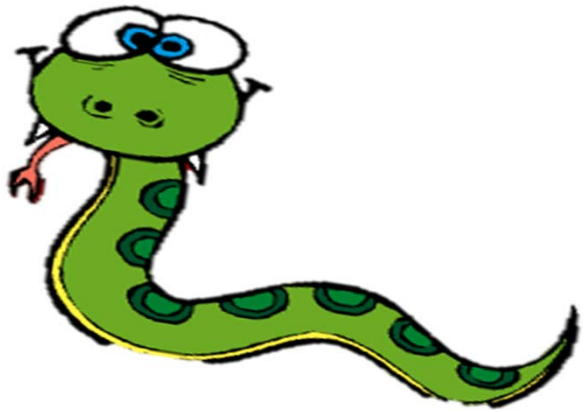
Python CET Course
Day 1



By Seow Khee Wei/Tan Kok Cheng

Republic Polytechnic

Source code: http://bit.ly/IPP_dec2019

Quick Survey

http://bit.ly/2Y8MTi2

Introduction of trainer

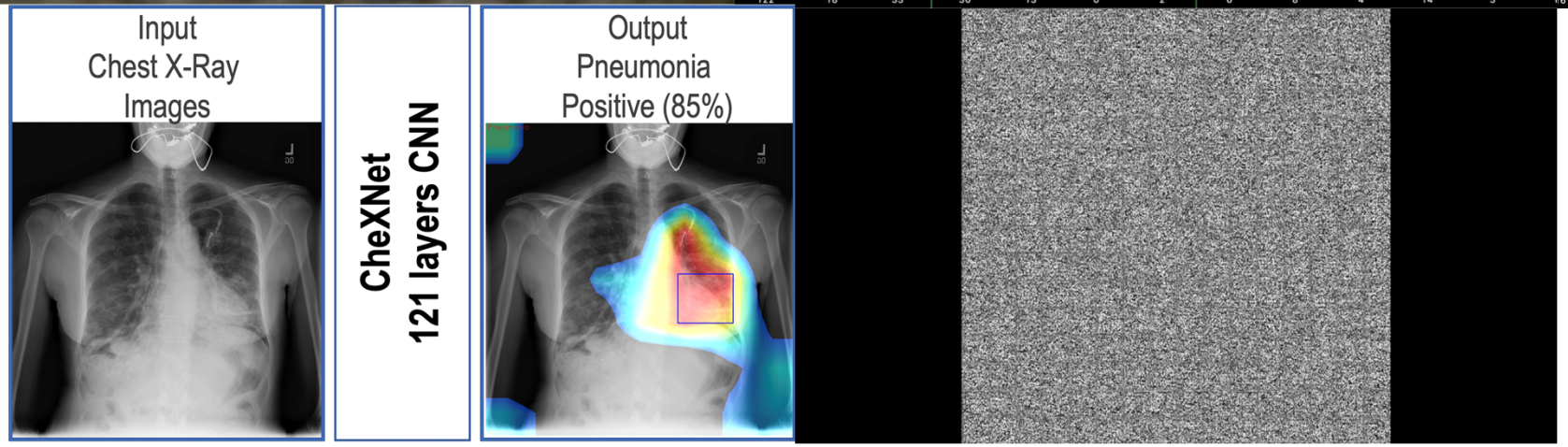
Name
Seow Khee Wei

Email
seow_khee_wei@rp.edu.sg

Telegram
@kwseow

Pet Project


pygame window
vase
Total count
Age
Age Groups
Male vs Female
Emotion
Emo Groups
Tracking|48|74|3517|8731|2019/01/21 21:42:18
CAPTURED FACES
VISITORS
AGE
EMOTIONS
TOTAL MALE FEMALE YOUTH YOUNG ADULT ADULT SENIOR ANGRY SCARED HAPPY SAD SURPRISED NEUTRAL
122 18 33 36 13 0 2 6 8 4 14 3 16


Input
Chest X-Ray
Images
CheXNet
121 layers CNN
Output
Pneumonia
Positive (85%)

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
- The tools and useful modules you can use to build your applications

Prereqs and Preparations

Before you attend this workshop, please make sure:

- Your laptop works
- You have installed the latest version of Python
- You have installed a suitable editor:
  We are using **Wing IDE Community Edition in this course**
- Usage of Chrome web browser

Programme Day One

Morning	Afternoon
• Install Python and using Wing 101 IDE • Data Types • If-else • For loops • Functions	• Try/except • String functions • String formatting • Writing a complete program • Graphical User Interface

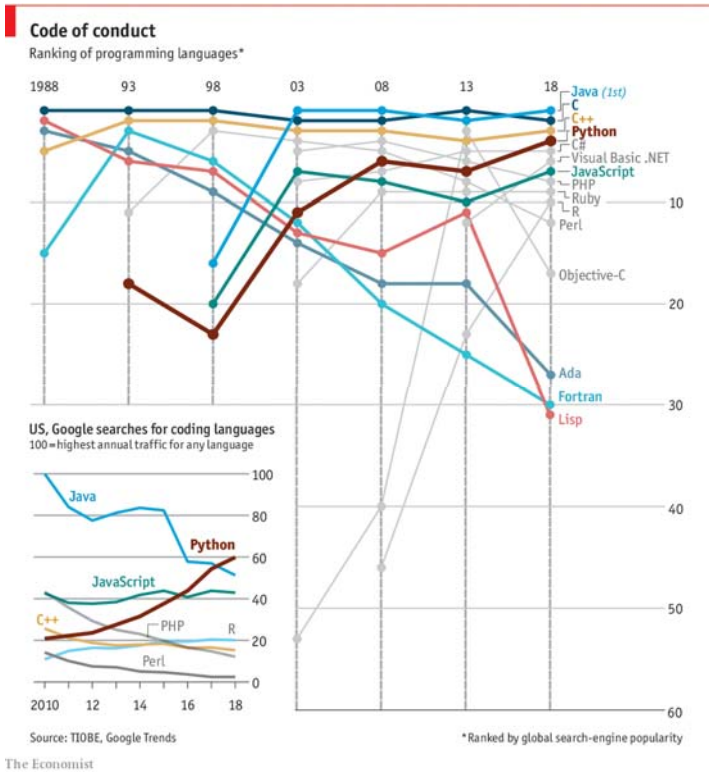
Programme Day Two

Morning	Afternoon
• Read and writing files • Copying, moving and deleting files and folders • Working with Excel • Processing CSV files	• Image Processing • Connecting to the Web • Sending emails • Telegram bot • Generating PDF • Creating Charts

Introduction to Python

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:
  - Codes 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

Introduction to Python

Interfaces to:

- COM, DCOM, ODBC
- Most databases (MySQL, MSQL, Sybase, Oracle,…)
- Java (Jpython)
- Many GUI / GFX libraries
  - Platform-independent: Tk, wxWindows, GTK, Pyglet
  - Platform-specific: MFC, MacOS, X11

Introduction to Python

Who uses Python?

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)

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

Introduction to Python

What is Python used for?

- Web development (Django, Flask, Google App Engine)
- File storage (Dropbox)
- Embedded in software packages (Maya, Blender)
- Scripting to perform simple (but mundane & repetitive) tasks

Introduction to Python

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

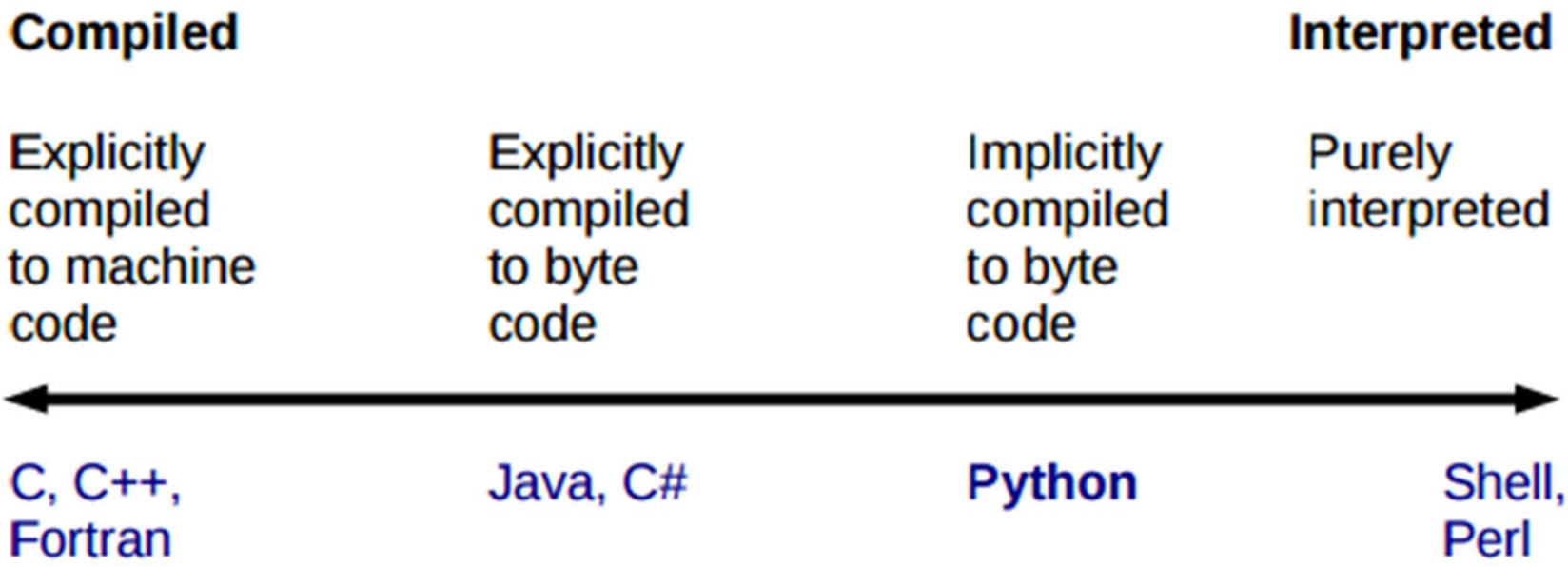
Introduction to Python

Release Date	Version	Release Date	Version
December, 1989	Implementation started	April 13, 1995	1.2
In the year 1990	Internal releases at CWI	October 13, 1995	1.3
February 20, 1991	0.9.0 (released to alt.sources)	October 25, 1996	1.4
February, 1991	0.9.1	January 3, 1998	1.5
Autumn, 1991	0.9.2	October 31, 1998	1.5.1
December 24, 1991	0.9.4	April 13, 1999	1.5.2
January 2, 1992	0.9.5 (Macintosh only)	September 5, 2000	1.6
April 6, 1992	0.9.6	October 16, 2000	2.0
Unknown, 1992	0.9.7beta	April 17, 2001	2.1
January 9, 1993	0.9.8	December 21, 2001	2.2
July 29, 1993	0.9.9	July 29, 2003	2.3
January 26, 1994	1.0.0	November 30, 2004	2.4
February 15, 1994	1.0.2	September 16, 2006	2.5
May 4, 1994	1.0.3	October 1, 2008	2.6
July 14, 1994	1.0.4	May 15th 2013	2.7.5
October 11, 1994	1.1	December 3, 2008	3.0
November 10, 1994	1.1.1	April 10th 2012	3.1.5
		May 15th, 2013	3.2.5

Python has two versions currently: 2.7.16 and 3.8.0

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

Where is Python on the Map



Python 2 vs. Python 3

- **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**

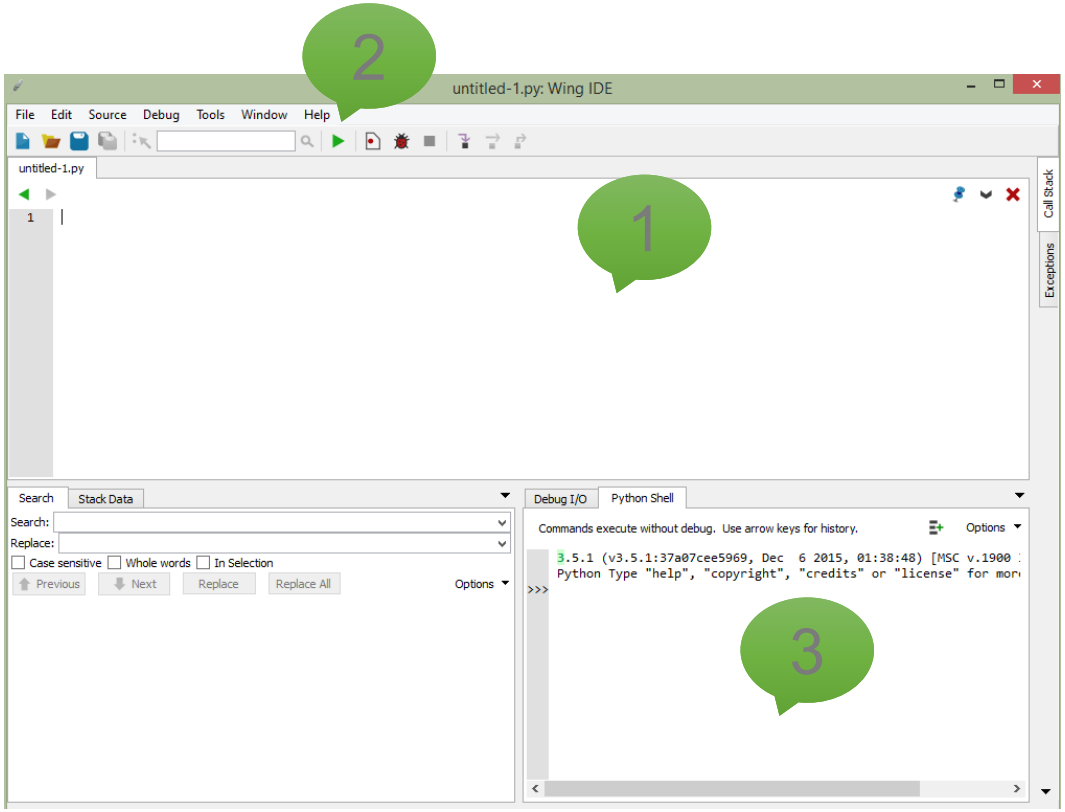
Why Python

- Focus on problem solving, and not on programming syntax

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

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

```
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);
    }
}
```

Run Wing101 IDE

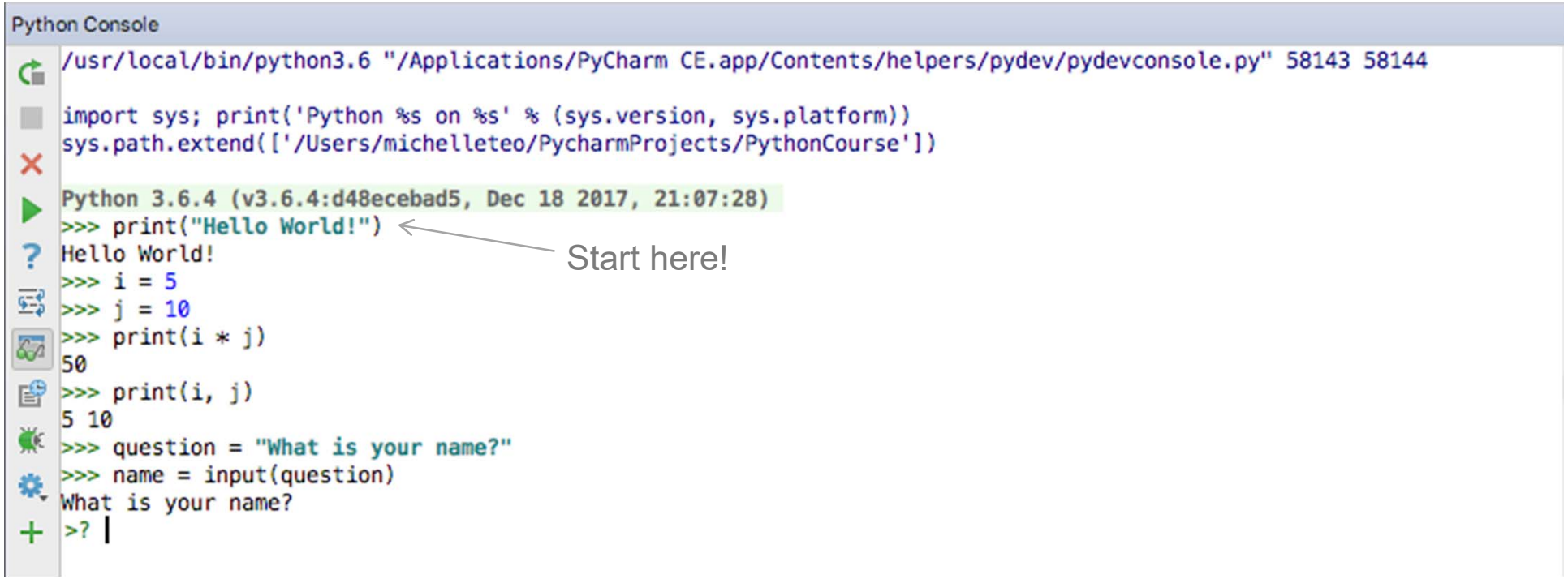


- Editor
- Run button
- Output window / Console

Using the Console

- Also known as the interpreter
- See the output straightaway
- Usually used to test very small chunks of code
- Type code after >>>
- Let’s try!

Using the Console



Data Types

We shall focus on these basic data types in our workshop:

Numbers

int	for whole numbers
float	for numbers with decimal point, e.g. 5.2, 2.0

Text

str	for a sequence of characters

Containers

list	a sequence of objects, use an index to access each object

Data Types – Numbers

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

- Python is a ‘friendly’ language
- The data type of a variable will change automatically depending on the values assigned to it.
- `i` is a variable
- In the 1st case, a float number is assigned to a variable `i`, so its data type is float.
- In the 2nd case, if an integer value is assigned to the same variable, its data type will change to integer.

Data Types – Strings

```
>>> s = "hello"
>>> t = "world"
>>> print(s+" "+t)
hello world
>>> s = "123"
>>> print(s+4)
Traceback (most recent call last):
  File "<string>", line 301, in runcode
  File "<interactive input>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
>>> print(s*4)
123123123123
>>>
```

Notes:
- For string assignment, you can use “hello” or ‘hello’
- But not “hello’

- Strings contain characters
- Strings can be added together with the + operator
- Strings can contain number characters, but they are not numbers (int or float)
- You cannot add a number to a string
- The * operator will replicate the string with an integer value

Data Types – Lists

```
>>> l = [1,2,3]
>>> print(l)
[1, 2, 3]
>>> print(l[0])
1
>>> print(l[2])
3
>>> print(l[3])
Traceback (most recent call last):
  File "<string>", line 301, in runcode
  File "<interactive input>", line 1, in <module>
IndexError: list index out of range
>>>
```

- Lists can contain anything
- Items in a list must be accessed by an index
- First index position starts from 0
- Python doesn't like it if you ask for something that is not in the list
- Try using a negative index, e.g. -1: What happens?

Data Types – Lists

```
>>> l = [1,2,3,4,5,6]
>>> print(len(l))
6
>>> print(l[2:5])
[3, 4, 5]
>>> print(l[:3])
[1, 2, 3]
>>> print(l[3:])
[4, 5, 6]
>>> print(l[-1])
6
>>>
```

n gives you the length or size of list

et a range of items using : colon
his is called slicing)

[:3]	first 3 items
[3:]	last 3 items

egative index gives you items from
e back

[-x]	x^{th} last item

Range

```
>>> 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

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`

Data Types – Conversion

```
>>> si="5"
>>> sf="5.5"
>>> print(int(si), float(sf))
5 5.5
>>> i=5
>>> print("int to str:"+str(i))
int to str:5
>>>
```

- You can convert anything into its string version:
  str(…)
- A string containing all digit characters a decimal point can be converted into a number type:
  int(…) or float(...)

Print Formatted Numbers

```
>>> import math
>>> print("Pi is " + str(math.pi))
Pi is 3.141592653589793
>>> print("Pi is approx %.2f"%(math.pi))
Pi is approx 3.14
>>> print("Pos or Neg: %+d %+d"%(-5,3))
Pos or Neg: -5 +3
>>>
```

Formatting numbers

d	int
%f	float

Special formatting

%.2f
: float
: two digits behind the point

%+d (or f)
: force print the sign

format-string.py

The time library

One of the functions in the time library is strftime, a flexible function to display the time based on certain format:

```
1 import time
2
3 print(time.strftime("Today is %d-%m-%Y %H:%M",time.localtime()))
```

https://docs.python.org/3/library/time.html?highlight=strftime#time.strftime

```
>>>
Today is 04-06-2017 17:30
>>>
```

Basic Arithmetic

Operator Name	Code	Example When x = 2 and 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 You must use * instead of x for multiplication.
x to the power of y	x ** y	x ** y means 2 to power of 1 and will give 2
Modulus	x % y	x % y will give 0 0 is the remainder from 2 divides by 1

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 Input

- You can use input() function to ask for user input.
- The 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'>
>>>
```

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.

```
Enter patient's name:-John
Enter patient's temperature:-37.5
John's temperature is 0.6 degree celsius from 36.9 degree celsius.
```

temperature_calculator.py

If-Else Statement

```
1  correct_password = "secret"
2  password = input("Enter password:-")
3
4  if password == correct_password:
5      print("You have entered correct password")
6  else:
7      print("You have entered wrong password")
```

```
Enter password:- secret
You have entered correct password
```

All lines, except line 5 are executed as **if password == correct_password** returns **True**.

```
Enter password:- letsguess
You have entered wrong password
```

All lines, except lines 6-7 are executed as **if password == correct_password** returns **False**.

password.py

Conditions in Decision Making

The condition(s) in a test can be expressed through the use of the following 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

if…elif…else

1. Type out the code below:

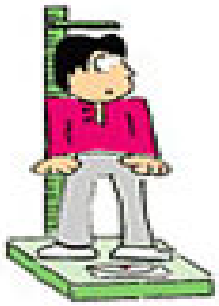
```
1   number1 = input("Enter first number:- ")
2   number2 = input("Enter second number:- ")
3   number1 = float(number1)
4   number2 = float(number2)
5
6   if number1 < number2:
7       print("First number is smaller than the second number.")
8   elif number1 > number2:
9       print("First number is greater than the second number.")
10  else:
11      print("Two numbers are equal.")
```

if-else-bigger-number.py

Exercise - BMI Calculator

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

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



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

bmi.py

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 Loops

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

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

for-loop.py

For Loops

```
>>> s = "freedom"
>>> print(s[:4])
free
>>> print(s[-3:])
dom
>>>
```

Slicing works for any sequence, so it works for strings too.

[:4] gets from the start till the fourth character

[-3:] gets the last third till the last character.

Data Types – Dictionary

```
{'year': '1995', 'type_of_public_transport': 'MRT', 'average_ridership': '740000'}
{'year': '1995', 'type_of_public_transport': 'LRT', 'average_ridership': '0'}
{'year': '1995', 'type_of_public_transport': 'Bus', 'average_ridership': '3009000'}
{'year': '1995', 'type_of_public_transport': 'Taxi', 'average_ridership': '0'}
{'year': '1996', 'type_of_public_transport': 'MRT', 'average_ridership': '850000'}
{'year': '1996', 'type_of_public_transport': 'LRT', 'average_ridership': '0'}
```

- A dictionary stores multiple key-value pairs
- E.g. In the first row of output, the dictionary contains 3 key-value pairs (which are the keys?)
- Every key is unique; no duplicate key within a dictionary
- A dictionary uses a set of curly brackets to store its key-value pairs {…}
  => Contrast with a list that uses square brackets to store its objects [...]
- To access a value in the dictionary, we use the key

Data Types – Dictionary

```
>>> scores = {'Mary': 90, 'Ben': 67, 'Jenny': 21}
>>> for s in scores:
...     print(s)
...
Mary
Ben
Jenny
```

- How does a `for` loop work on dictionaries?
- Doing '`for s in scores`' in the above code will assign the value of each key to `s`
- Change '`print(s)`' to '`print(s, scores[s])`', what do you get?

score-dictionary.py

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
Even #: 4
Odd #: 6
```

- Q: What if a user does not enter a positive integer?

odd-even1.py odd-even2.py

Functions

```
>>> def myFunction():
...     print("hello")
...
>>> type(myFunction)
<class 'function'>
>>> myFunction
<function myFunction at 0x03C74738>
>>> myFunction()
hello
>>>
```

```
>>> import math
>>> def calcPHI():
...     return (math.sqrt(5)-1)/2
...
>>> calcPHI()
0.6180339887498949
```

function-calcPHI.py

Functions

Passing parameters

- Python uses neither “pass-by-reference” nor “pass-by-value”
- It uses “pass-by-object”
- Arguments must be passed in order

```
>>> def multiply(a,b):
        return a*b

>>> multiply(2,3)
6
>>>
```

Alternatively, use parameter names to identify the arguments

```
>>> def identCar(car,colour):
...     print("Car %s has colour %s"%(car,colour))
...
>>> identCar(colour='red', car='honda')
Car honda has colour red
>>>
```

function-identCar.py

Default parameters

Default parameters values and checking if parameter has been passed

```
>>> def identCar(car=None,colour='red'):
...     if car == None:
...         print("You have to give me a car name")
...         return
...     print("Car %s has colour %s"%(car,colour))
...
>>> identCar(colour='blue')
You have to give me a car name
>>> identCar(car='toyota')
Car toyota has colour red
>>>
```

function-identCar.py

Arbitrary argument list

If you don't know how many parameters the function will receive, you can use *args, which will be a list.

```
>>> def addAll(*args):
...     sum=0
...     for num in args:
...         sum+=num
...     return sum
...
>>> addAll(1,2)
3
>>> addAll(1,2,3,4,5,6,7,8,9)
45
>>>
```

Create a function that takes in an unknown amount of parameters and returns the sum.

function-addAll.py

Function generator

```python
import math

def square_root(x):
    return math.sqrt(x)

result = square_root(4)
print (result)

square_root = lambda x: math.sqrt(x)

result = square_root(4)
print (result)
```

Output

```
C:\Python36\python.exe
2.0
2.0
```

Python has a special function generator called lambda.

This can be used to create templates functions.

```python
def make_adder(n):
    return lambda x: x + n

adder2 = make_adder(2)
val = adder2(5)
print (val)

adder3 = make_adder(4)
val = adder3(5)
print (val)
```

Output

```
C:\Python36\python.exe
7
9
```

function-generator.py

try .. except

Error handling is done through the use of exceptions that are caught in try blocks and handled in except blocks

```
>>> try:
...     5/0
... except Exception as e:
...     print("Exception ",type(e),": ",e.args)
...
Exception  <class 'ZeroDivisionError'> :  ('division by zero',)
>>>
```

```
>>> try:
...     5/0
... except:
...     print("error")
...
error
>>>
```

exception-divide.py

try .. except

You can also use the finally block. The code in the finally block will be executed regardless of whether an exception occurs.

```
>>> try:
...     5/0
... finally:
...     print("oeps, just before we run into an exception.")
...
oeps, just before we run into an exception.
Traceback (most recent call last):
  File "<string>", line 301, in runcode
  File "<interactive input>", line 2, in <module>
ZeroDivisionError: division by zero
>>>
```

exception-divide.py

try .. except

A good use for try expect is to check if the user has the specific library installed and if now, explains to the user what to do:

```
>>> try:
...     import special_module
... except ImportError:
...     print("Sorry, you don't have the special_module module installed,")
...     print("and this program relies on it.")
...     print("Please install or reconfigure special_module and try again.")
...
Sorry, you don't have the special_module module installed,
and this program relies on it.
Please install or reconfigure special_module and try again.
>>>
```

exception-import.py

try .. except

Another example is to check if a website is available:

```
1 from urllib.request import urlopen
2 def isOnline(reliableserver='http://www.google.com'):
3     try:
4         urlopen(reliableserver)
5         return True
6     except IOError:
7         return False
```

```
>>> isOnline()
True
>>>
```

String functions

Split

```
>>> a='python or java'
>>> b=a.split(' ')
>>> type(b)
<type 'list'>
>>> b
['python', 'or', 'java']
>>>
```

```
>>> a='python or java'
>>> b=a.split('on')
>>> b
['pyth', ' or java']
>>>
```

Join

```
>>> a=['python','and','java']
>>> b=' '.join(a)
>>> b
'python and java'
>>> c=','.join(a)
>>> c
'python,and,java'
>>>
```

string-function.py

Exercise – Find Longest Word

Create the function findLongestWord that takes in a sentence and returns the longest word. Hint: Use split()

longest-word.py

String formatting

Try this out yourself !

```
>>> import math
>>> a = math.pi
>>> a
3.141592653589793
>>> b=5
>>> c="python"
>>> line="%s %f %d"%(c,a,b)
>>> line
'python 3.141593 5'
>>>
```


```
>>> line="%03d"%(b)
>>> line
'005'
```

string-format1.py

Exercise - string formatting

Given the variable
I = “admin:$E*G$@R:/users/root:”

Can you print it like
User : admin
Password : $E*G$@R
Homedir : /users/root

string-format2.py

More string formatting

With c=”python”, a=3 and b=5

```
>>> "%-15s"%(c)
'python         '
>>>
```

```
>>> line="%15s %.0f %d"%(c,a,b)
>>> line
'         python 3 5'
>>>
```

```
>>> "%(language)s has %(#)03d quote types"%{'language':'python',"#":2}
'python has 002 quote types'
>>>
```

More about this string formatting technique can be found here:
http://docs.python.org/library/stdtypes.html#string-formatting-operations
format( )

Exercise – Xmas Tree

Question: Using string formatting and a loop, try to print the following xmas tree:

```
  #
 ###
#####
#######
```

string-xmas.py

The random library

random.randint(a, b)
Return a random integer N such that
a <= N <= b

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

Other random functions
random.shuffle(List)
random.choice(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.

Exercise - Guessing Game

- **Sample output**

```
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
```

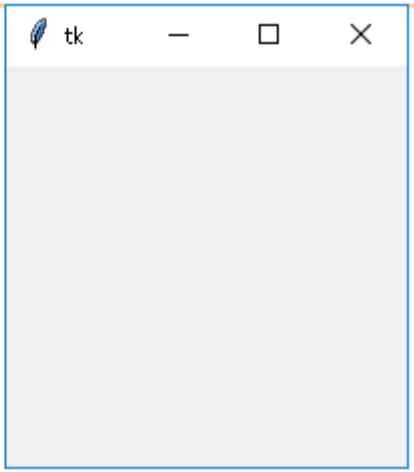
```
What is your name?
John
Well, John, I am thinking of a number between 1 and 20
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your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

Process finished with exit code 0
```

Graphical User Interface

https://wiki.python.org/moin/GuiProgramming

Tkinter – Python's standard GUI library
It is a commonly used GuiProgramming toolkit for Python.

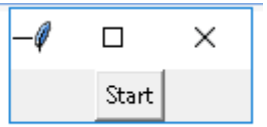
```
1 import tkinter
2
3 window = tkinter.Tk()
4 window.mainloop()
5
```



Graphical User Interface

Add a button

```
1  import tkinter
2
3  window = tkinter.Tk()
4
5  # Add a button
6  button1 = tkinter.Button(window, text="Start")
7  button1.pack()
8
9  window.mainloop()
10
```



Graphical User Interface

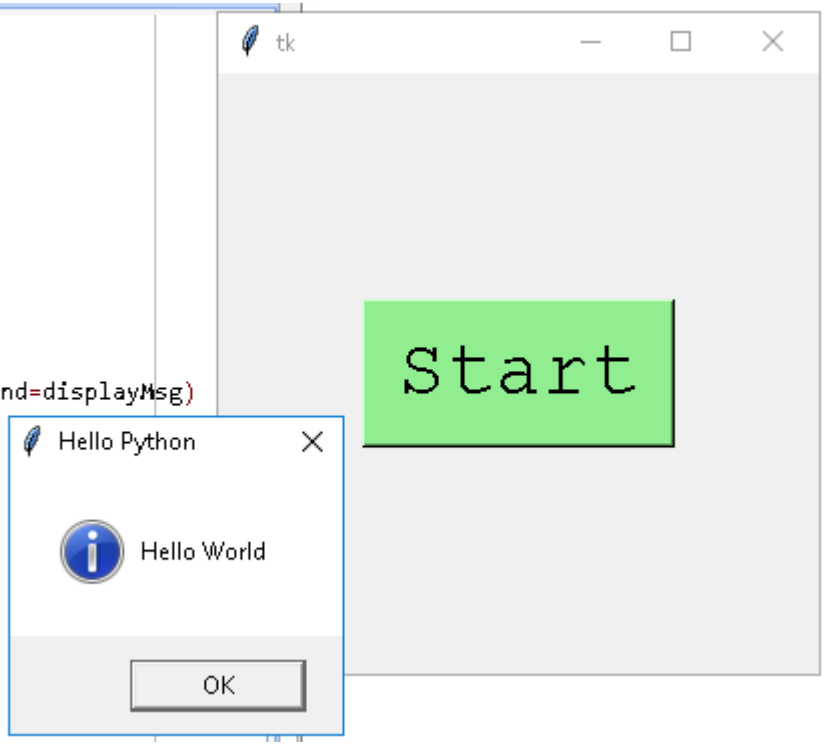
Set the window's size.
Configure the colour and position of the button.

```
1  import tkinter
2  import tkinter.messagebox
3
4  window = tkinter.Tk()
5
6  # Set the window's size
7  window.geometry("300x300")
8
9  # Add and configure a button
10 button1 = tkinter.Button(window, text="Start", bg="lightgreen")
11 button1.config(font=("Courier",30))
12 button1.pack(side="top", expand=tkinter.YES)
13
14 window.mainloop()
15
```

tk-button.py

Graphical User Interface

```python
import tkinter
import tkinter.messagebox

window = tkinter.Tk()

# Set the window's size
window.geometry("300x300")

def displayMsg():
    tkinter.messagebox.showinfo("Hello Python", "Hello World")

# Add and configure a button
button1 = tkinter.Button(window, text="Start", bg="lightgreen", command=displayMsg)
button1.config(font=("Courier",30))
button1.pack(side="top", expand=tkinter.YES)

window.mainloop()
```

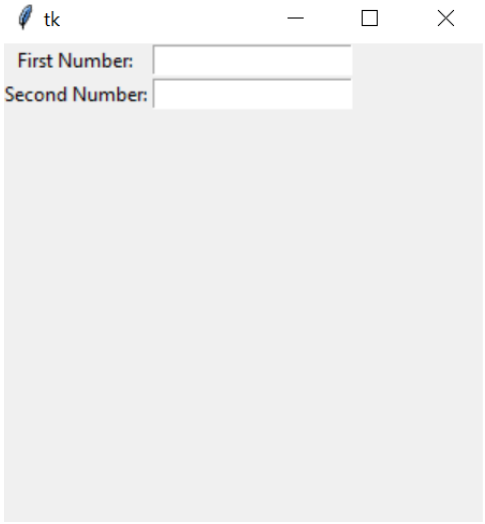


tk-button-messagebox.py

Graphical User Interface

- Specify position and size of UI components
  - Use grid to arrange component in row and column

```
from  tkinter  import  *

master  = Tk()
master .geometry  ("300x300"  )

l1 = Label (master , text ="First Number:")
l2 = Label (master , text ="Second Number:")
l1.grid (row =0, column =0)
l2.grid (row =1, column  =0)

e1  = Entry (master )
e2  = Entry (master )
e1.grid (row =0, column =1)
e2.grid (row =1, column =1)

mainloop  ( )
```

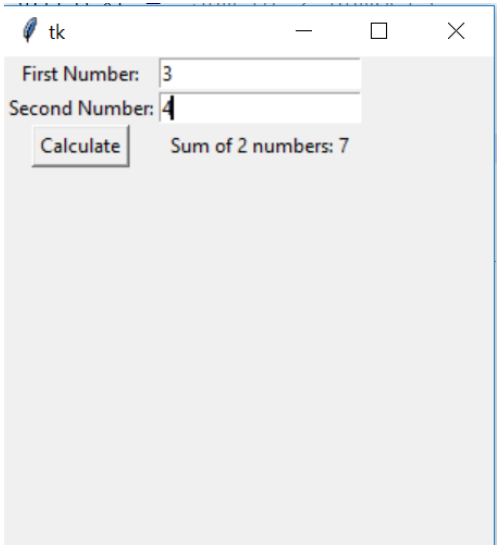


tk-grid.py

Graphical User Interface

- Getting input, and display result in label

```
from tkinter import *

def calculate():
    total = int(e1.get()) + int(e2.get())
    resultText = "Sum of 2 numbers: " + str(total)
    resultLabel.config(text=resultText)

master = Tk()
master.geometry("300x300")

l1 = Label(master, text="First Number:")
l2 = Label(master, text="Second Number:")
l1.grid(row=0, column=0)
l2.grid(row=1, column =0)

e1 = Entry(master)
e2 = Entry(master)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)

Button(master, text='Calculate',command=calculate).grid(row=2, column=0)
resultLabel = Label(master, text="Answer: ")
resultLabel.grid(row=2, column=1)

mainloop( )
```



tk-grid.py

Thank you