Introductory Programming Using Python

Day 1

By Tan Kok Cheng/ Seow Khee Wei

Republic Polytechnic

[eAttendance]

(SF) Introductory Programming using Python (04 - 05 Jun 2020)

https://forms.office.com/Pages/ResponsePage.aspx?id=0LCI9vB5pECGRDX83unQ82hg4AwDH9xMn406LA8StXhURVkyTENSUzgzNlJONjJERlVVN1hFNkhQQy4u

Click HERE for Live Attendance Report

Warm up!

https://bit.ly/kw_poll

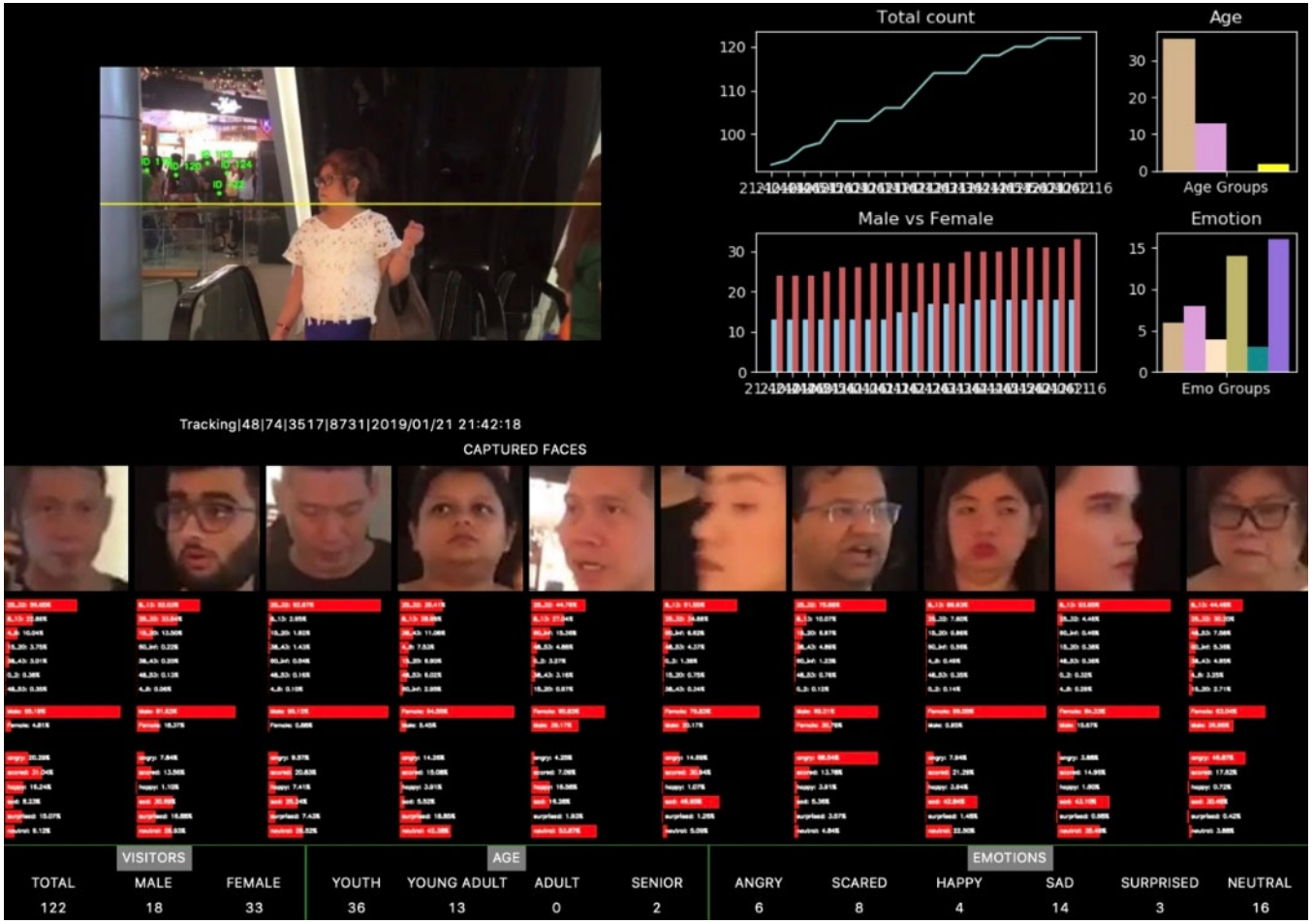
Introduction of trainer

Name
Seow Khee Wei

Telegram
@kwseow

Email
seow_khee_wei@rp.edu.sg

Projects


Total count
120
110
100
Age
30
20
10
0
Age Groups
Male vs Female
30
20
10
0
Emotion
15
10
5
0
Emo Groups
Tracking|48|74|3517|8731|2019/01/21 21:42:18
CAPTURED FACES
VISITORS
TOTAL
122
MALE
18
FEMALE
33
AGE
YOUTH
36
YOUNG ADULT
13
ADULT
0
SENIOR
2
EMOTIONS
ANGRY
6
SCARED
8
HAPPY
4
SAD
14
SURPRISED
3
NEUTRAL
16

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

Course Materials at bit.ly/rp-jun20

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

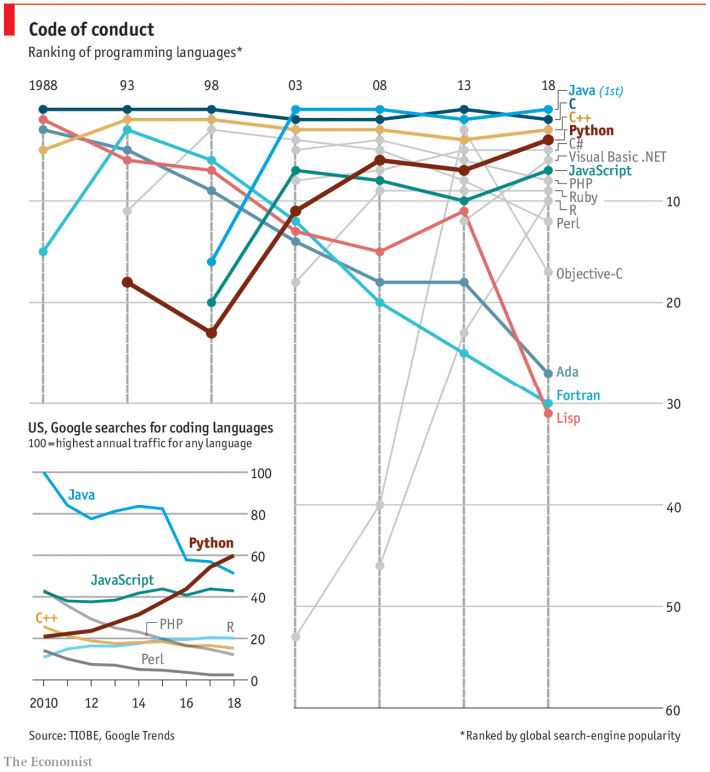
Programme Day Two

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

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

Who uses 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

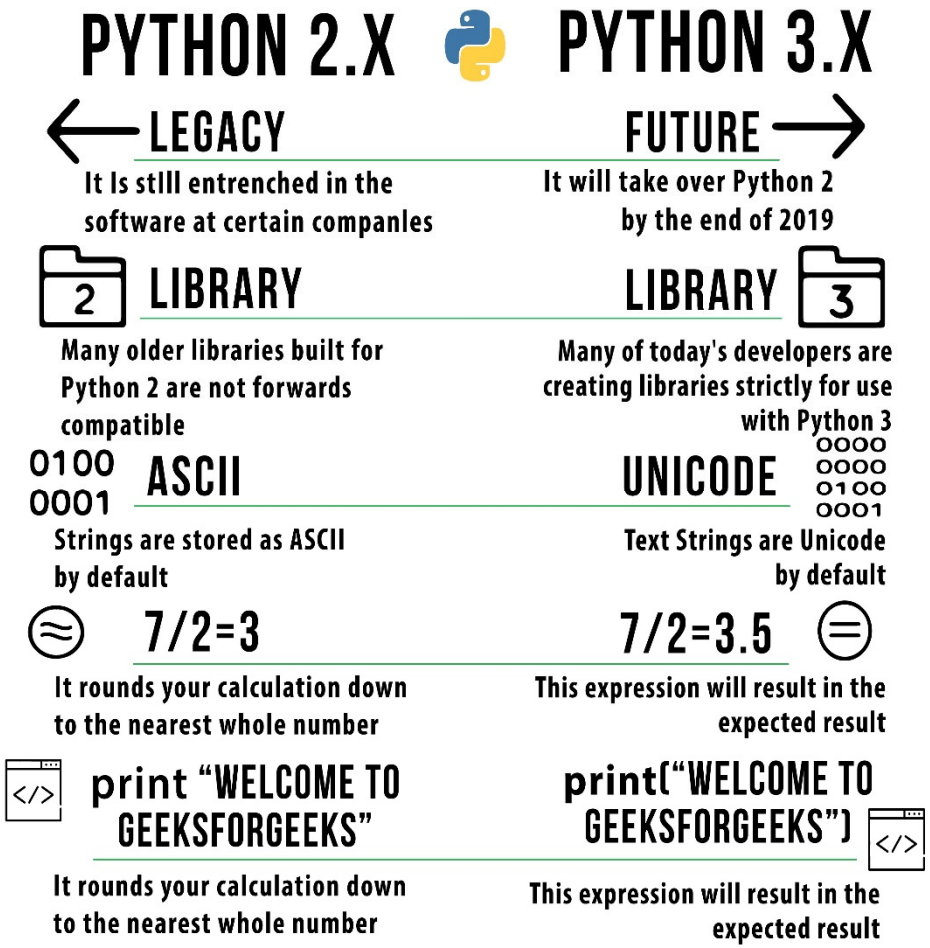
Who Invented Python

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

Python Versions

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



Python has two versions currently: 2.7.17 and 3.8.1

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

Python 2 reaches its End of Life (EOL) on 1 Jan 2020.

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**

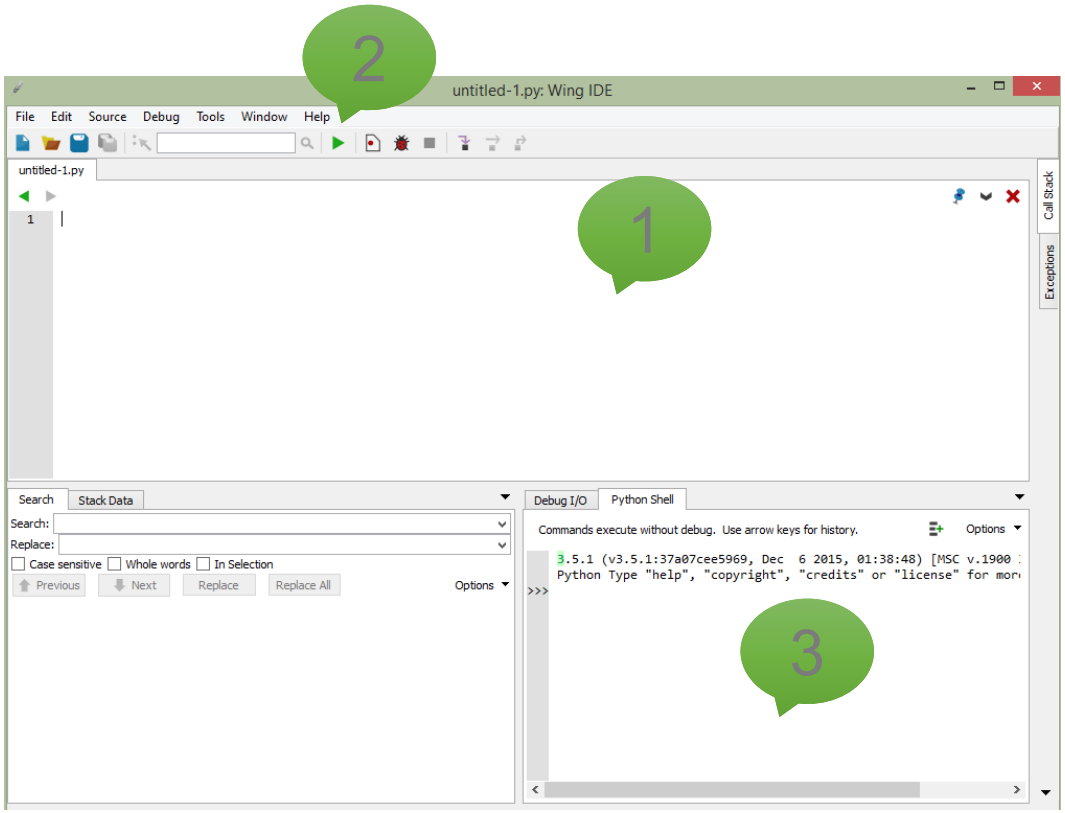
Why Python

- 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!

Interactive Python

- Let's do some simple mathematics with Python now!
- Run the following pieces of code in the python interpreter to see how effortlessly Python does it.

```
    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.
>>> 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, like no starting with a number.
  - E.g. Valid variable names: x, y, abc1234
  - Non-valid variable names: 1234abc
- To declare a variable to store a piece of data, simply assign a value to a name of your choice.
  - E.g. x = 100

Using variables?

- We can then use the variables in our codes
- To print out 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

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

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

Example	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

- You can add, subtract, multiply and divide numbers with numbers
  - 2 + 3
  - 2 * 3.0
  - 3/ 2
  - 2 – 6

- For String, take note of the following:
  - Add string to string
    - "hello" + "world" → "helloworld"
    - "3" + "5" → "35"
  - Multiply string with int
    - "x" * 5 → "xxxxx"
    - "x" * 5.5 → Error
  - Add string to numbers
    - "5" + 5 → Error

Common mistake:

```
age = 15
print ("age is " + age)
```

Correct method:

```
age = 15
print ("age is " + str(age))
```

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

Exercises

Example	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			

Lists

- In many other programming languages, arrays are used to store a collection of similar variables. Lists are Python's alternative for arrays.

- 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 out orange, not red! Take note that the index starts from 0.

Initializing Lists

- List elements are to be defined inside square brackets

```
>>> mylist1 = [10, 20, 30, 40]
>>> mylist2 = ["hello", 3.0, 5] # can mix different data type
>>> mylist3 = ["hello", 3.0, 5, [10, 20]] # nested list
```

Accessing List Elements

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

- Index starts with ‘0’ and ends with ‘length – 1’
- (-)ve indices, starting with -1 are used to refer to elements starting from the last.
- To find out how many elements are there in a list:

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

List Membership

- Check if an element exists in a list

```
>>> fruits = ['apple', 'orange', 'mango', 'banana', 'papaya']
>>> 'apple' in fruits
True
>>> 'book' in fruits
False
```

- Lists and for loops

```
>>> for fruit in fruits:
...     print("I like to eat " + fruit + "s!")
...
I like to eat apples!
I like to eat oranges!
I like to eat mangos!
I like to eat bananas!
I like to eat papayas!
```

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 Operation

- 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 5 from the list

5 mins

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

5 mins

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
```

10 mins

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.
```

10 mins

temperature_calculator.py

15 Mins Break

Selection/Decision Making

- An if-else statement is used in Python 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?

- The 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 will run.
- If it is of a value False, the the else portion of the code will execute.

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

True or False

- True and False are constants in Python
- Comparison Operators: ==, !=, <, <=, etc.

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

Boolean Logic Expressions

- You can also combine Boolean expresssion
  - true if a is true and b is true: a and b
  - true if a is true or b is true: a or b
  - true if a is false: not a

- Use parentheses as needed to disambiguate complex Boolean expressions.

```
if i == 0 and not a <= 5 or b == 7:
    do something...

        if (i == 0 and (not a <= 5)) or b == 7:
            do something...
```

Selection/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!")
```

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

Mini Quiz

- 1 >= 1 or 1 == 0
- 1 == 1 and 1 == 0
- 1 == 1 and 1 == 0 or 1 == 1
- 1 == 1 and not 1 == 0
- not 1 == 0 and 1 == 1
- 1 != 1 or 1 = 0

5 mins

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")
```

Exercise

Write the code to ask a user to enter his favorite sports. Print the result as given in the table below:

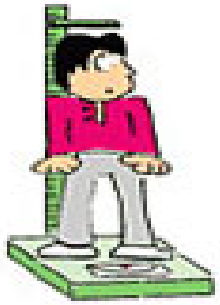
Input	Display or Print the following
Soccer	Your Favorite Sport is Soccer
Basketball	Your Favorite Sport is Basketball
Badminton	Your Favorite Sport is Badminton

10 mins

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

15 mins

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.

Range

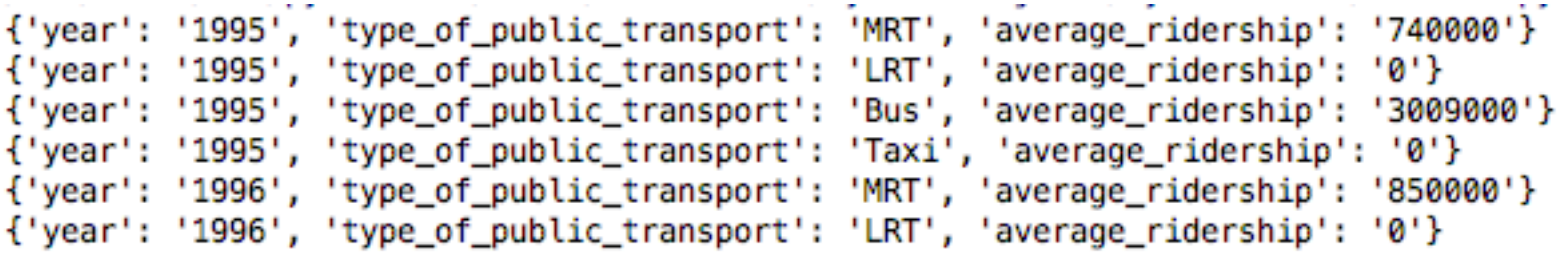
```
>>> print(list(range(10)))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> print(list(range(1,10)))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> print(list(range(1,10,2)))
[1, 3, 5, 7, 9]
>>>
>>> print(list(range(10,1,-1)))
[10, 9, 8, 7, 6, 5, 4, 3, 2]
>>>
```

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 – 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?

15 mins

odd-even1.py odd-even2.py

Lunch

60 Mins Break

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
>>>
```

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 Function **def**

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

Function definition begins with '**def**'.

Function name and its arguments.

```
def get_answer(filename):
    "Documentation String"
    line1
    line2
    return answer
```

The indentation matters…

The keyword '**return**' indicates the value to be sent back to the caller.

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: Defining and Calling a Function

- The syntax for a function call is:

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

Returning value from Function

- Compare this 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_ara(width , height ):
    print (width * height )
```

print a value

- Most function should return instead of print a value

Example: Function with return value

- **Function to calculate area of rectangle**

Define the function

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

Using the function

```
area1 = cal_area  ( 4, 5)
area2 = cal_area  (2, 3)
total_area = area1 + area2
print("Total area: " + sr(total_area))
```

Function overloading? No.

- **No function overloading in Python**
  - Two different functions can't have the same name, even if they have different arguments.

```
>>> def add(a, b):
        return a+n

>>> def add(a, b, c):
        return a+b+c

>>> add(1,2)

Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    add(1,2)
TypeError: add() takes exactly 3 arguments (2 given)
```

Example: Define and Use Function

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

Argument list

```
def getBiggerNumber(num1, num2):
    if num1 > num2:
        return num1
    else:
        return num2
```

Exercises

- Write a function that takes in a number as argument, and returns that number

- Write a function that takes in a number as argument, and returns that number incremented by 1

- Write a function that calculates and returns the double of the number given as argument

10 mins

Exercises

- 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.

```
>>> 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

15 mins

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

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 **15 mins**

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
  l = “admin:$E*G$@R:/users/root:”

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

string-format2.py

10 mins

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

20 mins

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.

20 mins

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
```

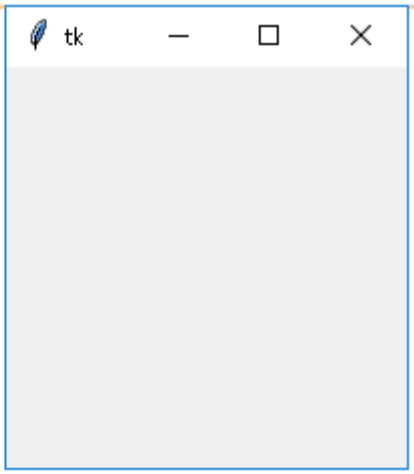
```
What is your name?
John
Well, John, I am thinking of a number between 1 and 20
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your guess is too high.
Take a guess
10
Your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

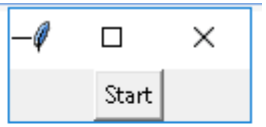
```
1 import tkinter
2
3 window = tkinter.Tk()
4 window.mainloop()
5
```



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
```



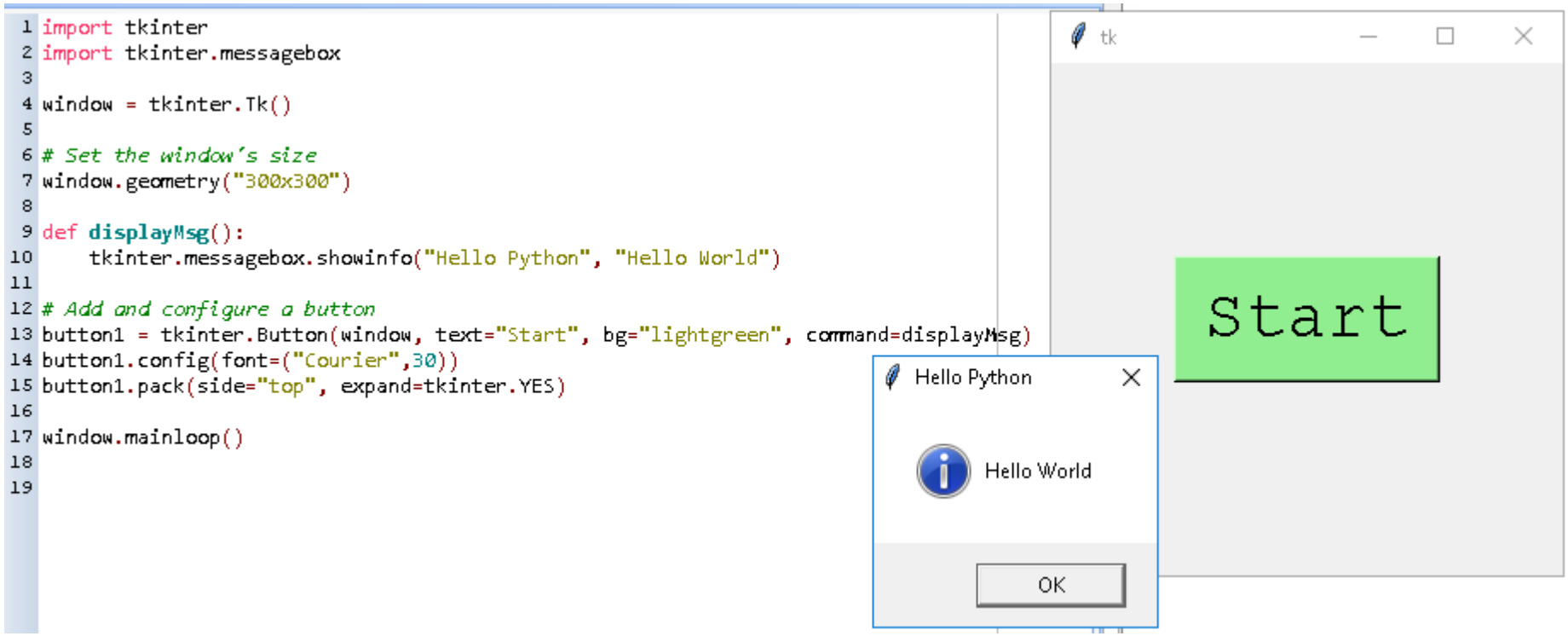
Graphical User Interface

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



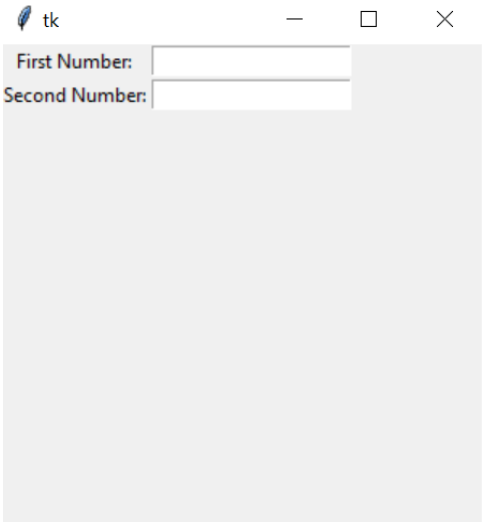
```
import tkinter
import tkinter.messagebox

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 ( )
```

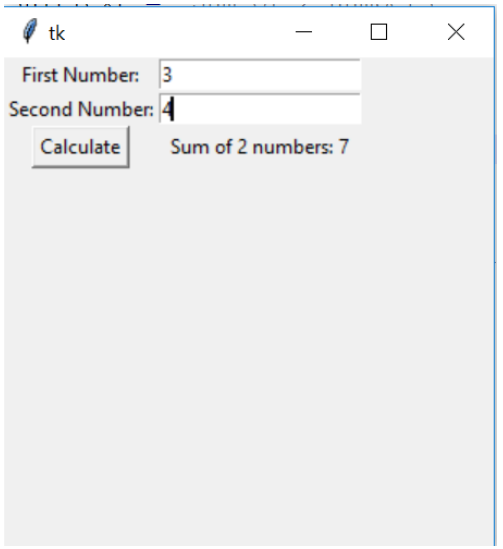


tk-grid.py

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

Let’s compete!

https://bit.ly/kw_poll

SCAN ME

Thank you

Solution to Data Types Exercises

Example	Variable Name	Data Type	Value
weight = 65.5	weight	float	65.5
gpa = 3	gpa	Int	3
gender = “Female”	gender	str	“Female”
Enabled = False	Enabled	bool	False
height = 180 + 5.0	height	float	185.0
w = float(4) + 3	w	float	7.0
x = 7/2	x	float	3.5
y = int(4.5) + 5.0	y	float	9.0
z = str(“1”) * 4	z	str	“1111”

Mini Quiz Solutions

- 1 >= 1 or 1 == 0 — True
- 1 == 1 and 1 == 0 — False
- 1 == 1 and 1 == 0 or 1 == 1 — True
- 1 == 1 and not 1 == 0 — True
- not 1 == 0 and 1 == 1 — True
- 1 != 1 or 1 = 0 — Error