# Python

Trainer: Mirza Touseef

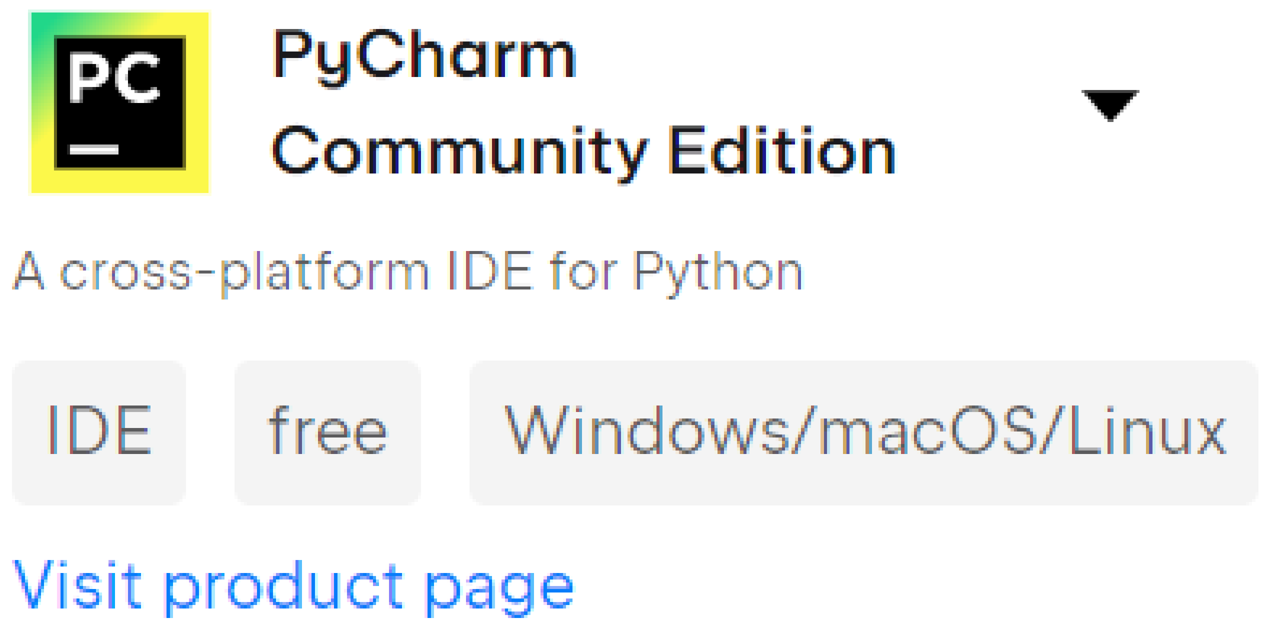
**Riphah Institute of System Engineering**

# Contents

## Why should you learn to write programs?

# Python 3

- **Python 3** is a newer version of the Python programming language which was released in December 2008
- It is **backward incompatible**
- https://www.python.org/



PyCharm
Community Edition

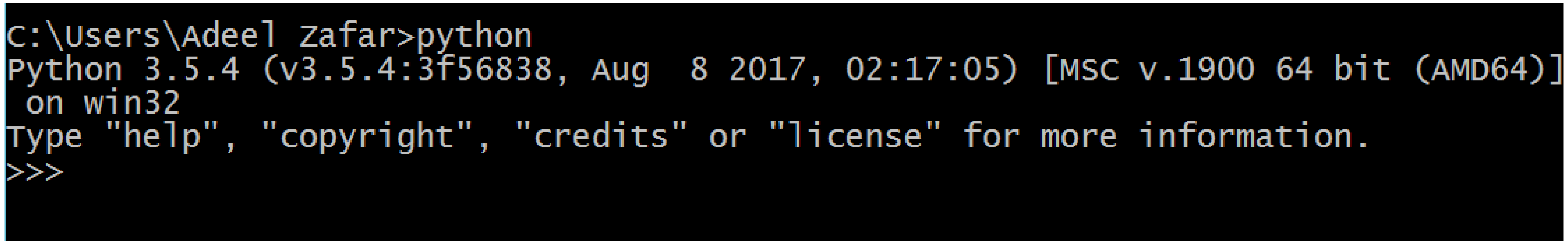
A cross-platform IDE for Python

IDE | free | Windows/macOS/Linux

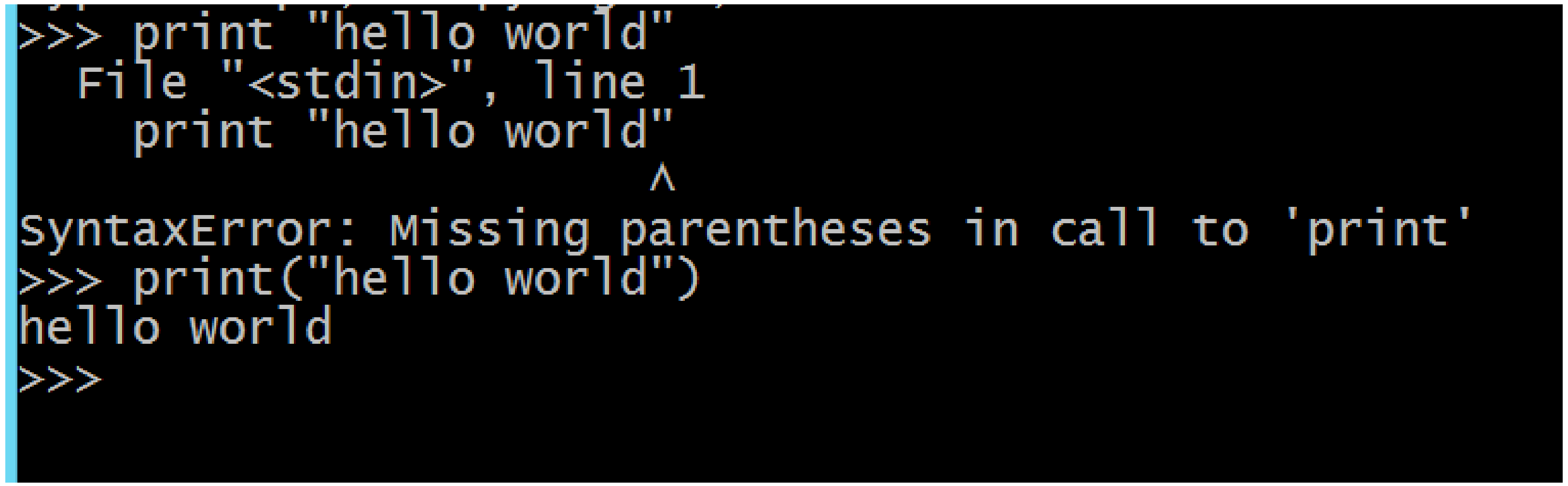
Visit product page

## Activity-1

- Install Python and Text Editor of your own choice

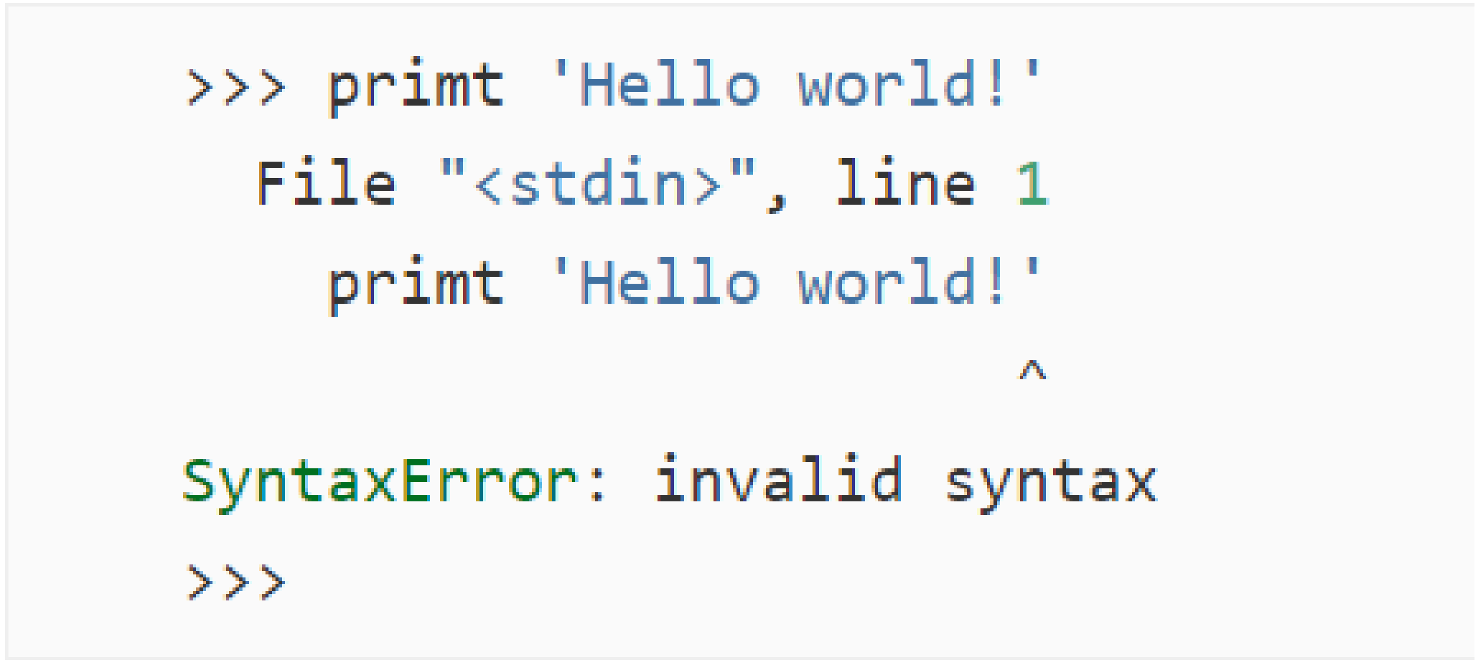


```
C:\Users\Adeel Zafar>python
Python 3.5.4 (v3.5.4:3f56838, Aug  8 2017, 02:17:05) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



```
>>> print "hello world"
  File "<stdin>", line 1
    print "hello world"
                      ^
SyntaxError: Missing parentheses in call to 'print'
>>> print("hello world")
hello world
>>>
```

## What is wrong with the following code:



```
>>> primt 'Hello world!'
  File "<stdin>", line 1
    primt 'Hello world!'
                       ^
SyntaxError: invalid syntax
>>>
```

## What will the following program print out:

```python
x = 43
x = x + 1
print(x)
```

## Values and types

- A value is one of the basic things a program works with, like a letter or a number

- The values we have seen so far are 1, 2, and “Hello, World!”

- These values belong to different types:
  - 2 is an integer,
  - and “Hello, World!” is a string

## Values and types

```
>>> type('Hello, World!')
<class 'str'>
>>> type(17)
<class 'int'>


>>> type(3.2)
<class 'float'>
```

## Variables

- One of the most powerful features of a programming language is the ability to manipulate variables

- A variable is a name that refers to a value.

- An assignment statement creates new variables and gives them values

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897931
```

## Variables

- The type of a variable is the type of the value it refers to

```
>>> type(message)
<class 'str'>
>>> type(n)
<class 'int'>
>>> type(pi)
<class 'float'>
```

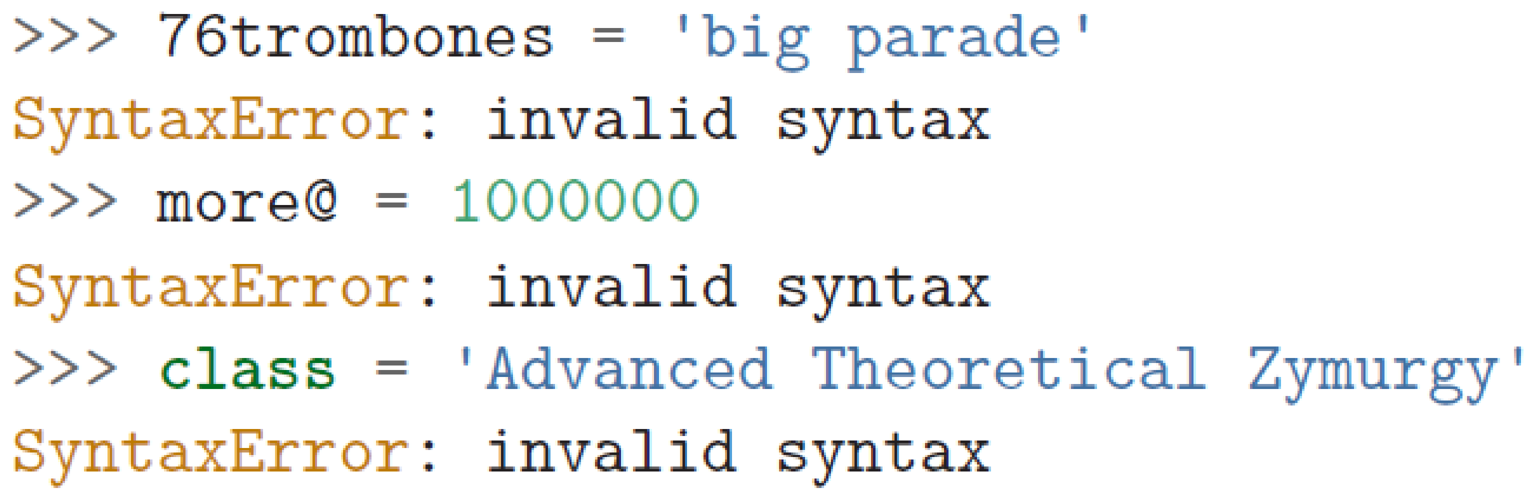
## Variable names and keywords

- Programmers generally choose names for their variables that are meaningful and document what the variable is used for

- Variable names can be arbitrarily long. They can contain both letters and numbers, but they **cannot start with a number**

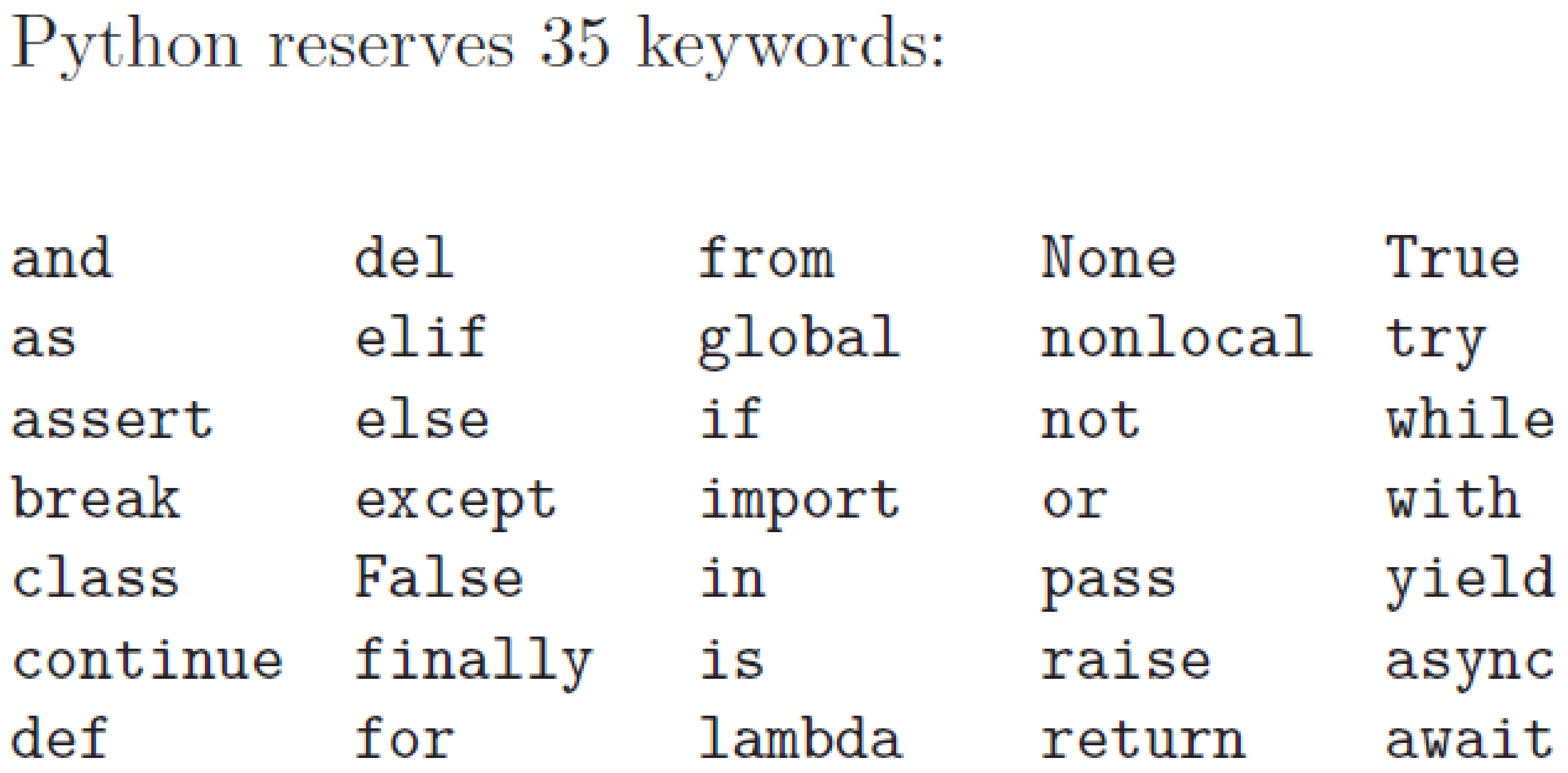
- It is legal to use uppercase letters, but it is a good idea to begin variable names with a lowercase letter

## Variable names and keywords



```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

## Reserved Words



Python reserves 35 keywords:

| | | | | |
|---|---|---|---|---|
| and | del | from | None | True |
| as | elif | global | nonlocal | try |
| assert | else | if | not | while |
| break | except | import | or | with |
| class | False | in | pass | yield |
| continue | finally | is | raise | async |
| def | for | lambda | return | await |

## Statements

- A statement is a unit of code that the Python interpreter can execute

- We have seen two kinds of statements: **print being an expression statement and assignment**

- When you type a statement in interactive mode, the interpreter executes it and displays the result, if there is one

## Script

- A script usually contains a sequence of statements

- If there is more than one statement, the results appear one at a time as the statements execute

```
print(1)
x = 2
print(x)
```

produces the output

```
1
2
```

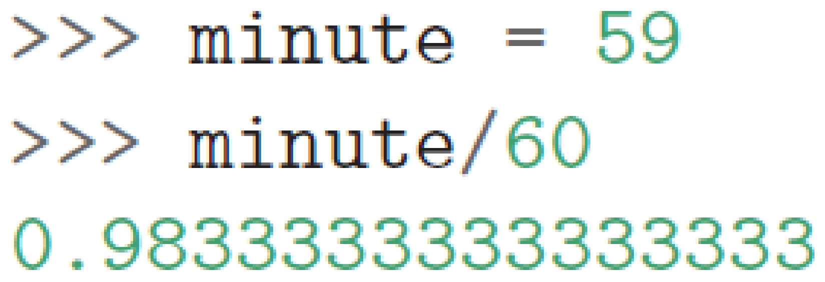
## Operators and operands

- Operators are special symbols that represent computations like addition and multiplication

- The values the operator is applied to are called operands

- The operators +, -, *, /, and ** perform addition, subtraction, multiplication, division, and exponentiation, as in the following examples

## Operators and operands



```
>>> minute = 59
>>> minute/60
0.9833333333333333
```

## Expressions

- An expression is a combination of values, variables, and operators
- A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions

```
17
x
x + 17
```

## Activity - 2

- **Type the following statements in the Python interpreter to see what they do:**

```
5
x = 5
x + 1
```

# Operator Precedence Rules

Highest precedence rule to lowest precedence rule:

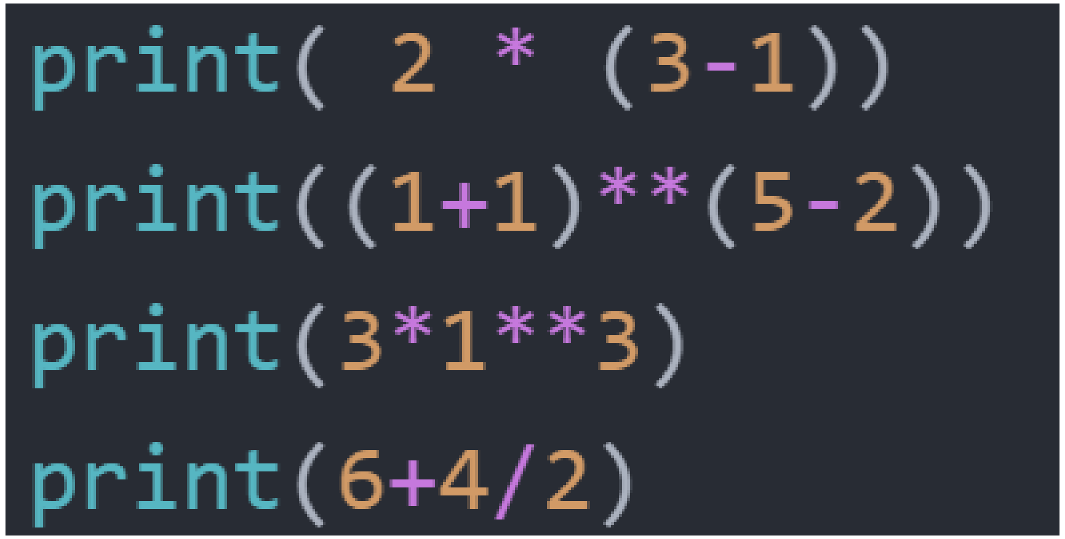
| | |
|---|---|
| Parentheses are always respected | Parenthesis |
| Exponentiation (raise to a power) | Power |
| Multiplication, Division, and Remainder | Multiplication |
| Addition and Subtraction | Addition |
| Left to right | Left to Right |

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
```

Parenthesis
Power
Multiplication
Addition
Left to Right

$$\begin{tikzcd}
2\ **\ 3 \arrow[d] \\
8\ /\ 4 \arrow[d] \\
2\ *\ 5 \arrow[d] \\
1\ +\ 10 \arrow[d] \\
11
\end{tikzcd}$$

## Order of operations



```
print( 2 * (3-1))
print((1+1)**(5-2))
print(3*1**3)
print(6+4/2)
```

## Modulus operator

- The modulus operator works on integers and yields the remainder when the first operand is divided by the second

- In Python, the modulus operator is a percent sign (%)

## Modulus operator

```
>>> quotient = 7 // 3
>>> print(quotient)
2
>>> remainder = 7 % 3
>>> print(remainder)
1
```

## Modulus operator

- The modulus operator turns out to be surprisingly useful
- For example, you can check whether one number is divisible by another: if x % y is zero, then x is divisible by y

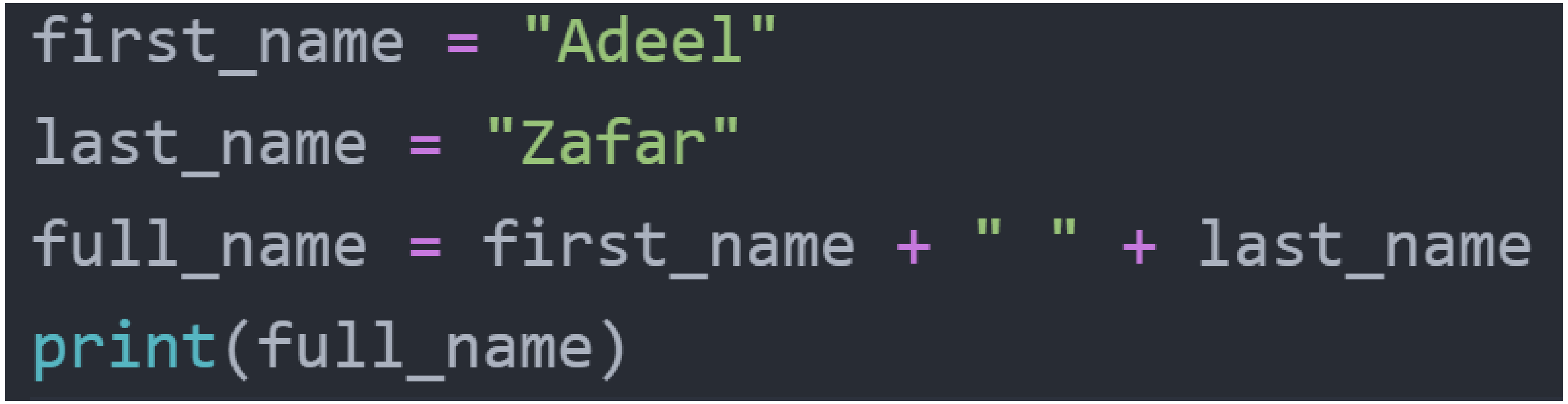
## Modulus operator

- You can also extract the right-most digit or digits from a number
- For example, x % 10 yields the right-most digit of x (in base 10). Similarly, x % 100 yields the last two digits

## String operations

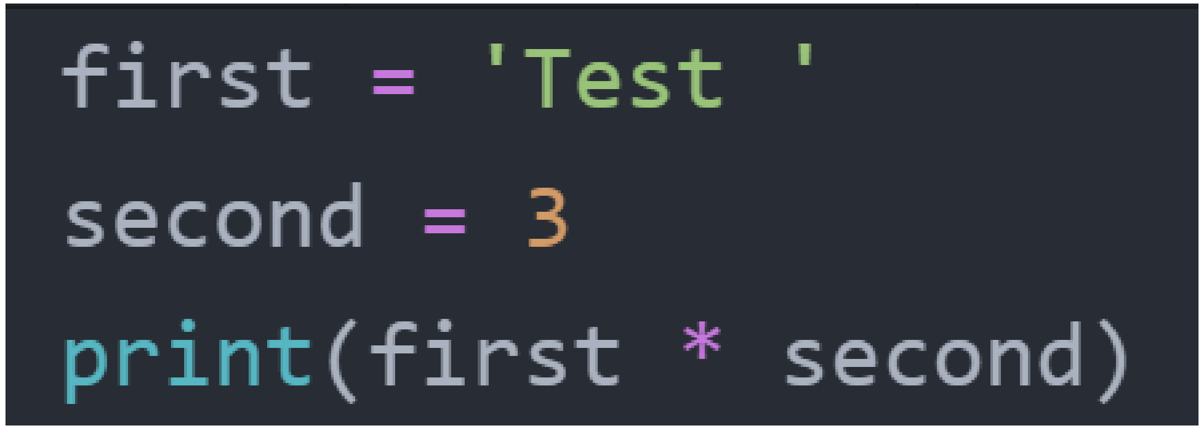
- The + operator works with strings, but it is not addition in the mathematical sense
- Instead it performs concatenation, which means joining the strings by linking them end to end

## String operations



```python
first_name = "Adeel"
last_name = "Zafar"
full_name = first_name + " " + last_name
print(full_name)
```

## String operations

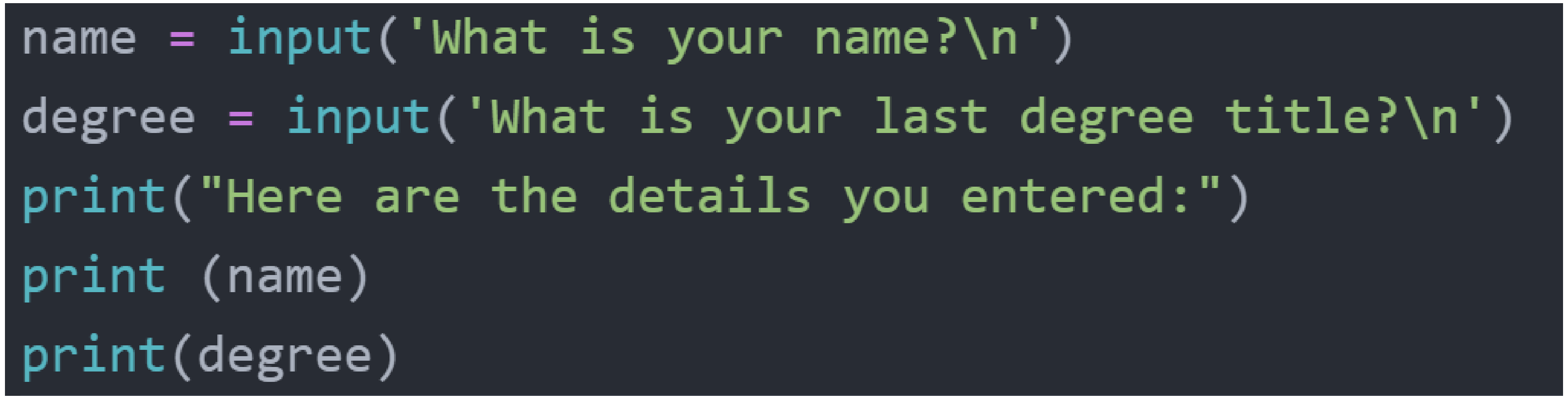


```
first = 'Test '
second = 3
print(first * second)
```

## Asking the user for input

- Sometimes we would like to take the value for a variable from the user via their keyboard
- Python provides a built-in function called **input** that gets input from the keyboard1
- When this function is called, the program stops and waits for the user to type something
- When the user presses Return or Enter, the program resumes and input returns what the user typed as a string

## Asking the user for input



```
name = input('What is your name?\n')
degree = input('What is your last degree title?\n')
print("Here are the details you entered:")
print (name)
print(degree)
```

## Escape Sequences

- The sequence \n at the end of the prompt represents a *newline, which is a special* character that causes a line break

- That’s why the user’s input appears below the prompt

## Conversion

- If you expect the user to type an integer, you can try to convert the return value to int using the int() function

## Comments

- As programs get bigger and more complicated, they get more difficult to read
- Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why
- For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing
- These notes are called comments, and in Python they start with the # symbol:

```
# compute the percentage of the hour that has elapsed
```

## Choosing mnemonic variable names

- A memory aid
- We often give variables mnemonic names to help us remember what is stored in the variable

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

```
hours = 35.0
rate = 12.50
pay = hours * rate
print(pay)
```

```
x1q3z9ahd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ahd * x1q3z9afd
print(x1q3p9afd)
```

## Debugging

- At this point, the syntax error you are most likely to make is an illegal variable name, like class and yield, which are keywords, or odd~job and US$, which contain illegal characters

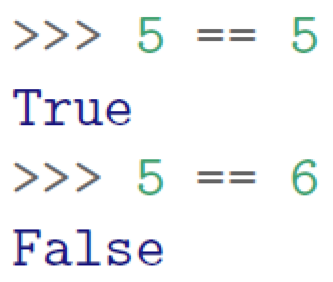
## Activity - 3

- **Write a program that uses input to prompt a user for their name and then welcomes them**
- **Write a program to prompt the user for hours and rate per hour to compute gross pay**
- **Write a program which prompts the user for a Celsius temperature, convert the temperature to Fahrenheit, and print out the converted temperature**

## Conditional execution : Boolean expressions

- A boolean expression is an expression that is either true or false

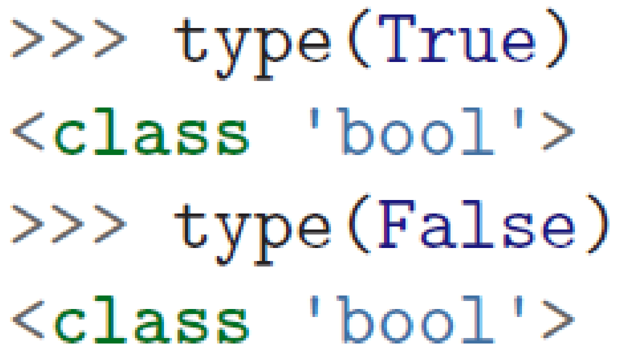
- The following examples use the operator ==, which compares two operands and produces True if they are equal and False otherwise:



```
>>> 5 == 5
True
>>> 5 == 6
False
```

## Conditional execution : Boolean expressions

- True and False are special values that belong to the class bool; they are not strings:



```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

## Conditional execution : Boolean expressions

```
x != y               # x is not equal to y
x > y                # x is greater than y
x < y                # x is less than y
x >= y               # x is greater than or equal to y
x <= y               # x is less than or equal to y
x is y               # x is the same as y
x is not y           # x is not the same as y
```

## Logical operators

- There are three logical operators: and, or, and not

- The semantics (meaning) of these operators is similar to their meaning in English

- For example,

x > 0 and x < 10

is true only if x is greater than 0 and less than 10

## Logical operators

- n%2 == 0 or n%3 == 0 is true if *either of the conditions is true, that is, if the* number is divisible by 2 *or 3*

- Finally, the not operator negates a boolean expression, so not (x > y) is true if x > y is false; that is, if x is less than or equal to y

## Conditional execution

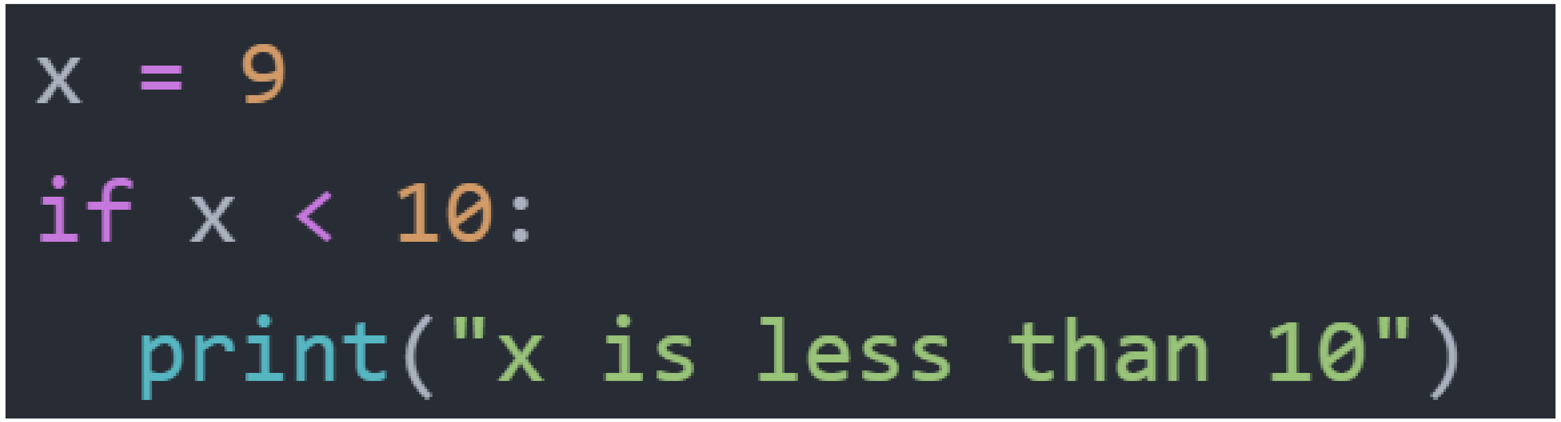
- In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly

- Conditional statements give us this ability. The simplest form is the if statement

## Conditional execution

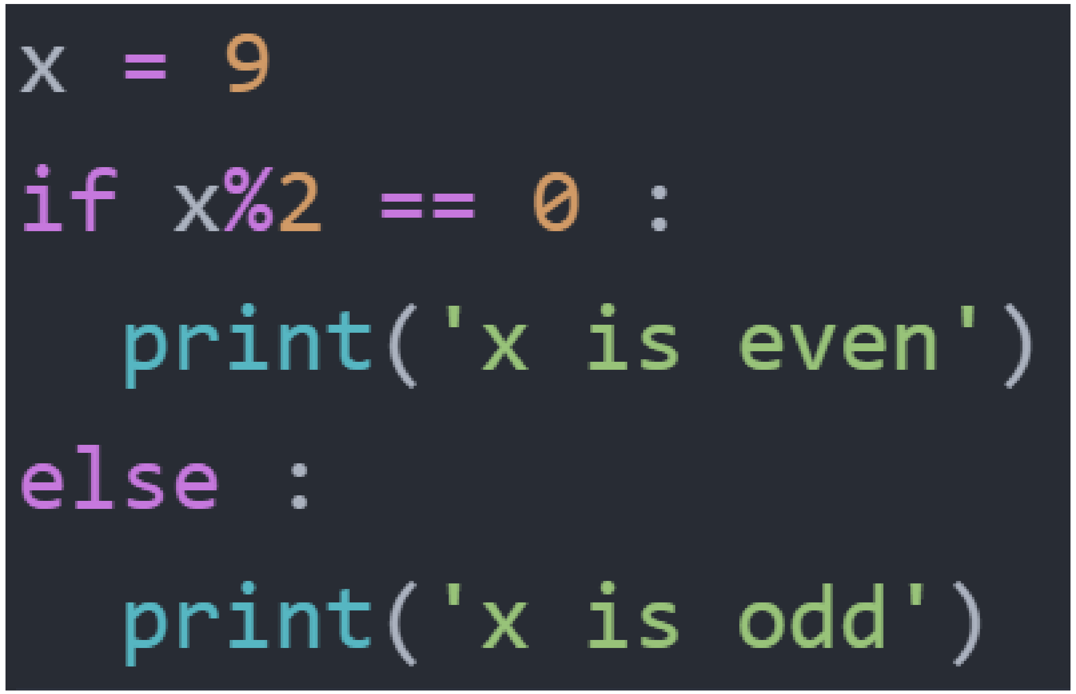
- if statements have the same structure as function definitions or for loops
- The statement consists of a header line that ends with the colon character (:) followed by an indented block
- Statements like this are called compound statements because they stretch across more than one line

## Conditional execution



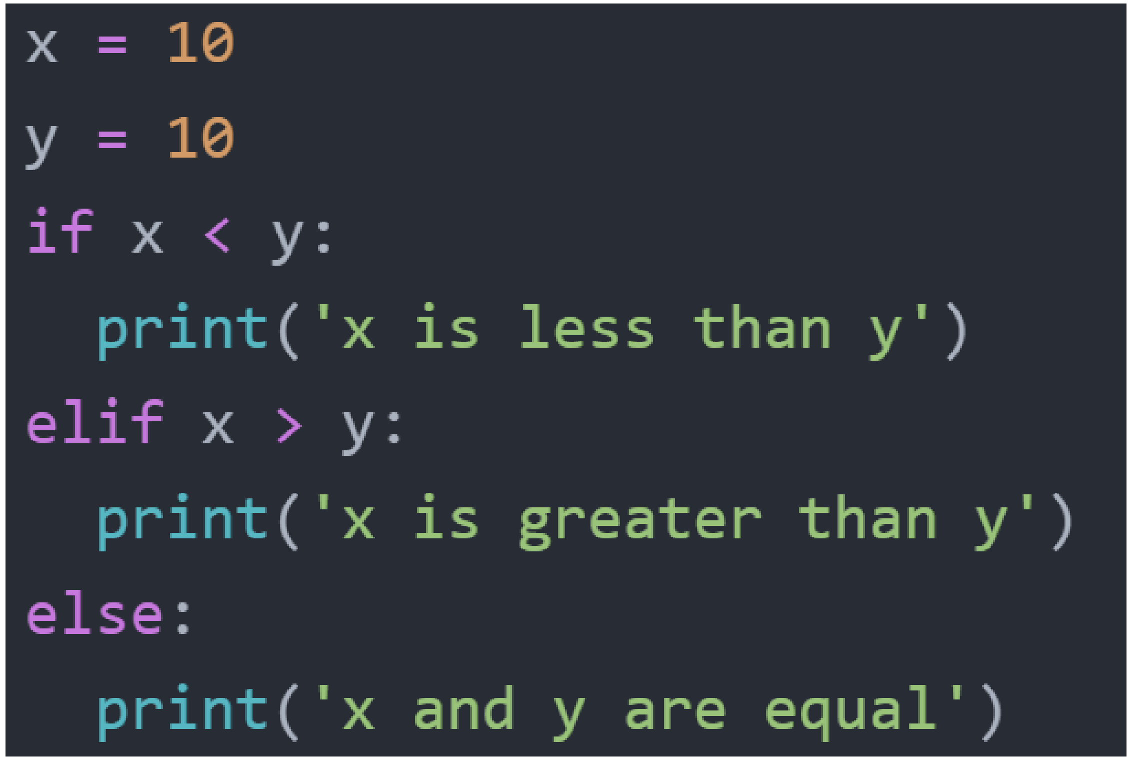
```
x = 9
if x < 10:
  print("x is less than 10")
```

## Alternative execution


```
x = 9
if x%2 == 0 :
  print('x is even')
else :
  print('x is odd')
```

## Chained conditionals

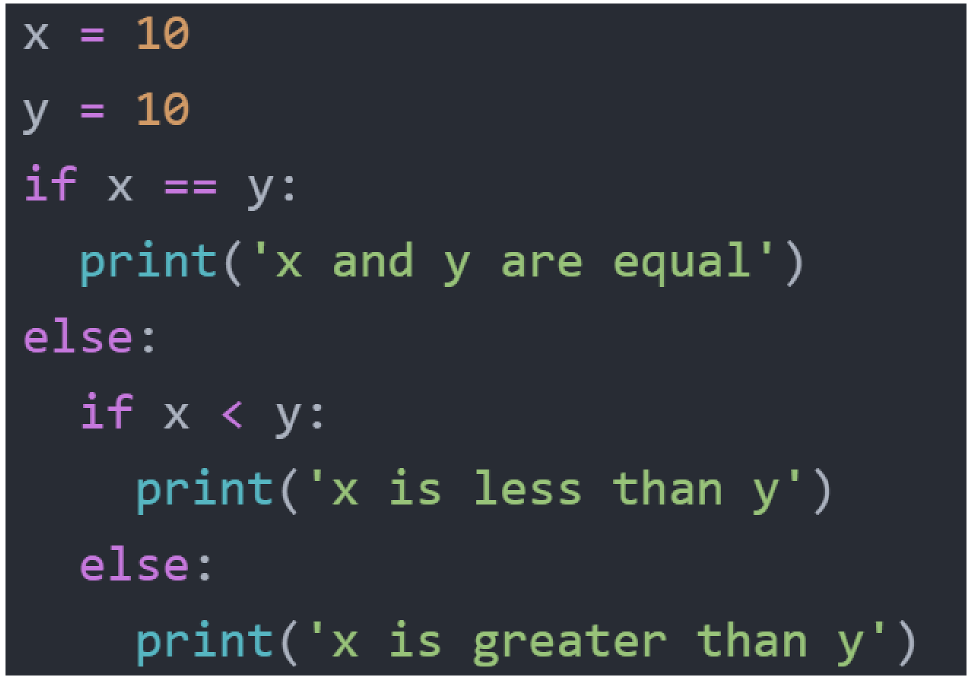
- Sometimes there are more than two possibilities and we need more than two branches



```python
x = 10
y = 10
if x < y:
  print('x is less than y')
elif x > y:
  print('x is greater than y')
else:
  print('x and y are equal')
```

# Nested conditionals

- One conditional can also be nested within another. We could have written the three-branch example like this



```python
x = 10
y = 10
if x == y:
  print('x and y are equal')
else:
  if x < y:
    print('x is less than y')
  else:
    print('x is greater than y')
```

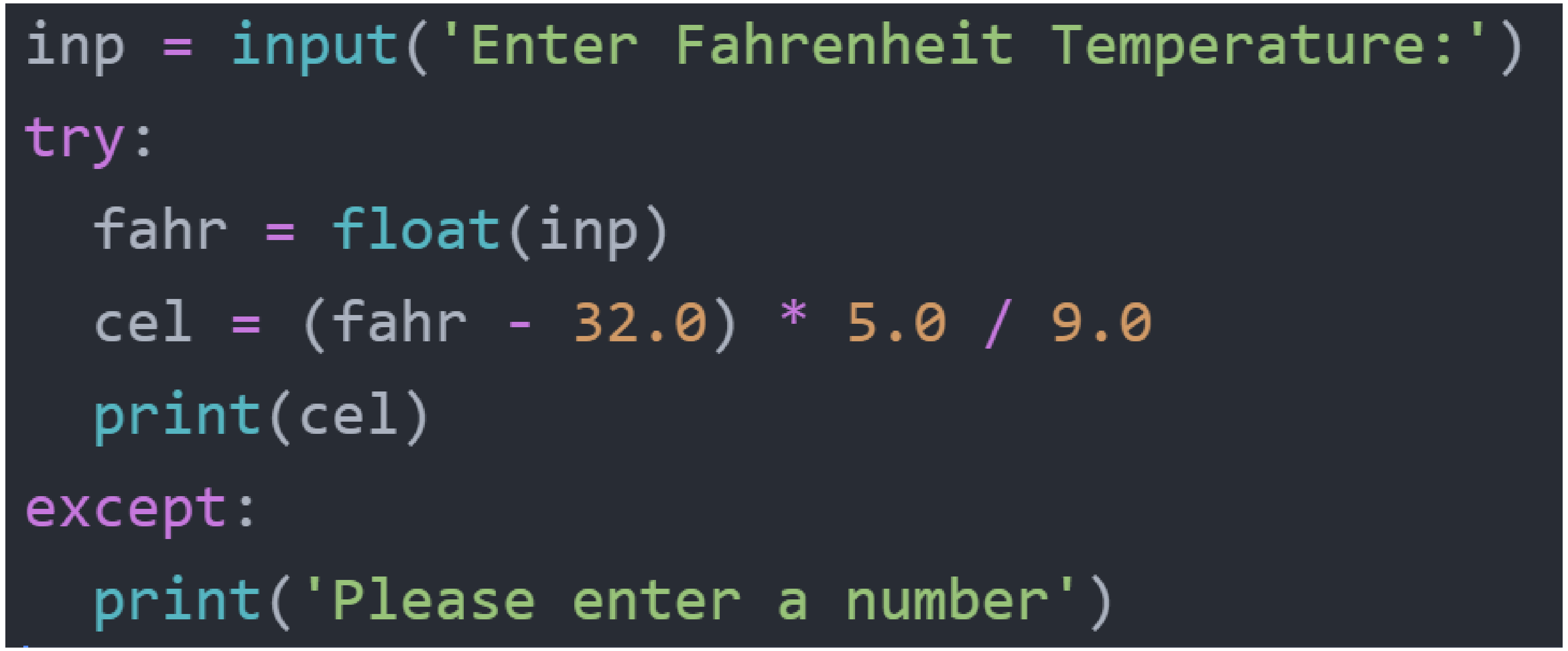
## Catching exceptions using try and except

- There is a conditional execution structure built into Python to handle these types of expected and unexpected errors called “try / except”

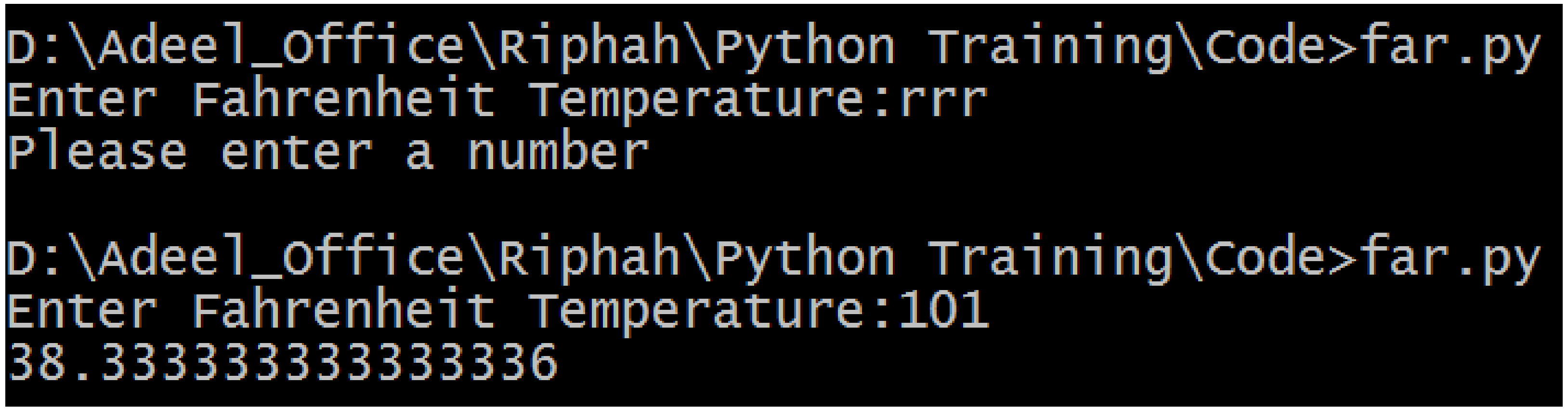
- The idea of try and except is that you know that some sequence of instruction(s) may have a problem and you want to add some statements to be executed if an error occurs

- These extra statements (the except block) are ignored if there is no error

## Catching exceptions using try and except



```python
inp = input('Enter Fahrenheit Temperature:')
try:
  fahr = float(inp)
  cel = (fahr - 32.0) * 5.0 / 9.0
  print(cel)
except:
  print('Please enter a number')
```



```
D:\Adeel_Office\Riphah\Python Training\Code>far.py
Enter Fahrenheit Temperature:rrr
Please enter a number

D:\Adeel_Office\Riphah\Python Training\Code>far.py
Enter Fahrenheit Temperature:101
38.333333333333336
```

## Short-circuit evaluation of logical expressions

```
>>> x = 6
>>> y = 2
>>> x >= 2 and (x/y) > 2
True
>>> x = 1
>>> y = 0
>>> x >= 2 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and (x/y) > 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

## Debugging

- The traceback Python displays when an error occurs contains a lot of information, but it can be overwhelming

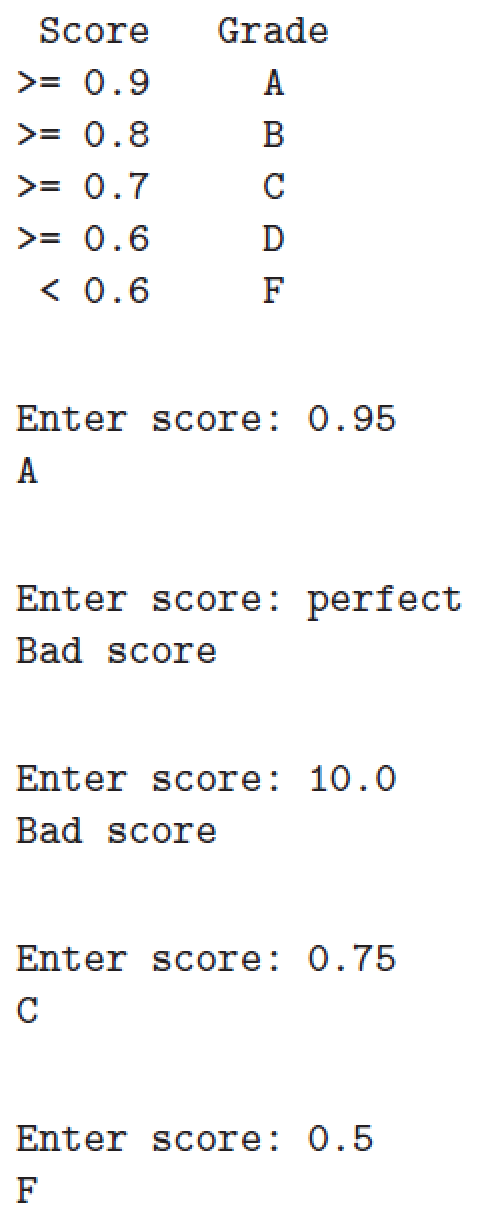
- The most useful parts are usually:
  - What kind of error it was, and
  - Where it occurred.

## Debugging

- Syntax errors are usually easy to find, but there are a few gotchas
- Whitespace errors can be tricky because spaces and tabs are invisible and we are used to ignoring them

# Activity-4

- **Write a program to prompt for a score between 0.0 and 1.0. If the score is out of range, print an error message. If the score is between 0.0 and 1.0, print a grade using the following table:**



```
 Score   Grade
>= 0.9     A
>= 0.8     B
>= 0.7     C
>= 0.6     D
 < 0.6     F


Enter score: 0.95
A


Enter score: perfect
Bad score


Enter score: 10.0
Bad score


Enter score: 0.75
C


Enter score: 0.5
F
```

## Activity - 5

- Write a program to prompt the user for hours and rate per hour using input to compute gross pay. Pay the hourly rate for the hours up to 40 and 1.5 times the hourly rate for all hours worked above 40 hours. Use 45 hours and a rate of 10.50 per hour to test the program (the pay should be 498.75). You should use **input** to read a string and **float()** to convert the string to a number. Do not worry about error checking the user input - assume the user types numbers properly

## Activity - 5

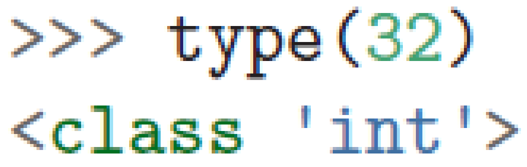
- Rewrite your pay program using try and except so that your program handles non-numeric input gracefully by printing a message and exiting the program. The following shows executions of the program
  - Enter Hours: forty
  - Error, please enter numeric input

# Functions : Function calls

- In the context of programming, a function is a named sequence of statements that performs a computation

- When you define a function, you specify the name and the sequence of statements

- Later, you can “call” the function by name



```
>>> type(32)
<class 'int'>
```

## Functions

- The name of the function is type
- The expression in parentheses is called the argument of the function
- The argument is a value or variable that we are passing into the function as input to the function

## Functions

- The result, for the type function, is the type of the argument

- It is common to say that a function “takes” an argument and “returns” a result

- The result is called the return value

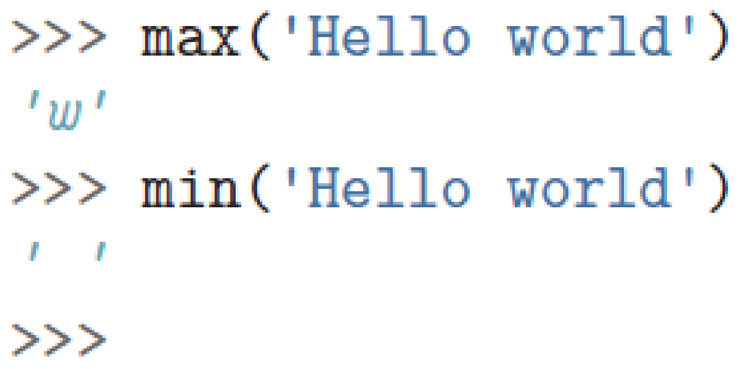
## Built-in functions

- Python provides a number of important built-in functions that we can use without needing to provide the function definition

- The creators of Python wrote a set of functions to solve common problems and included them in Python for us to use

# Built-in functions : Max () – Min ()

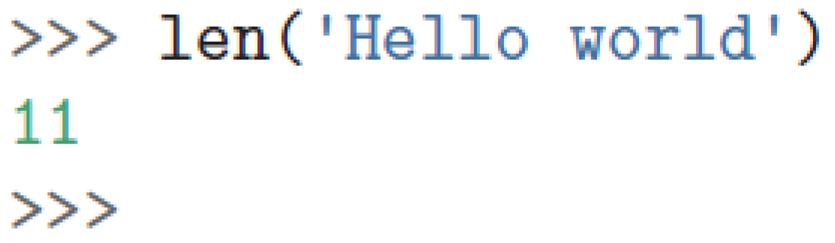
- The max and min functions give us the largest and smallest values in a list, respectively:



```
>>> max('Hello world')
'w'
>>> min('Hello world')
' '
>>>
```

## Built-in functions : len()

- Another very common built-in function is the len function which tells us how many items are in its argument
- If the argument to len is a string, it returns the number of characters in the string

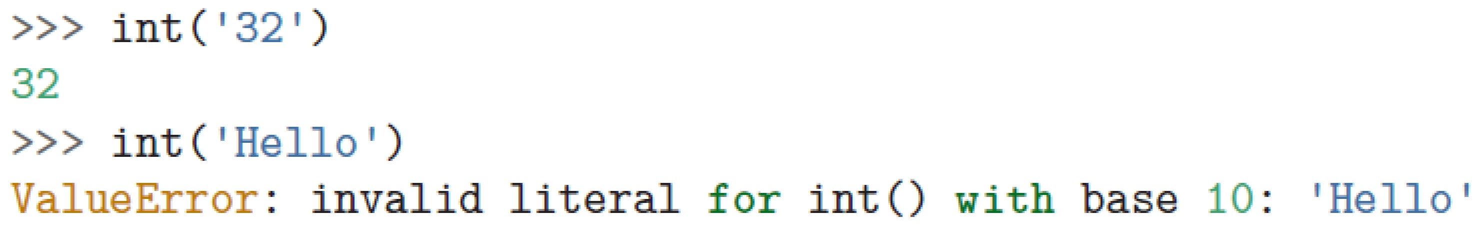


```
>>> len('Hello world')
11
>>>
```

# Type conversion functions

- Python also provides built-in functions that convert values from one type to another

- The int function takes any value and converts it to an integer, if it can, or complains otherwise:



```
>>> int('32')
32
>>> int('Hello')
ValueError: invalid literal for int() with base 10: 'Hello'
```

# Type conversion functions

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

```
>>> float(32)
32.0
>>> float('3.14159')
3.14159
```

```
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

## Math functions

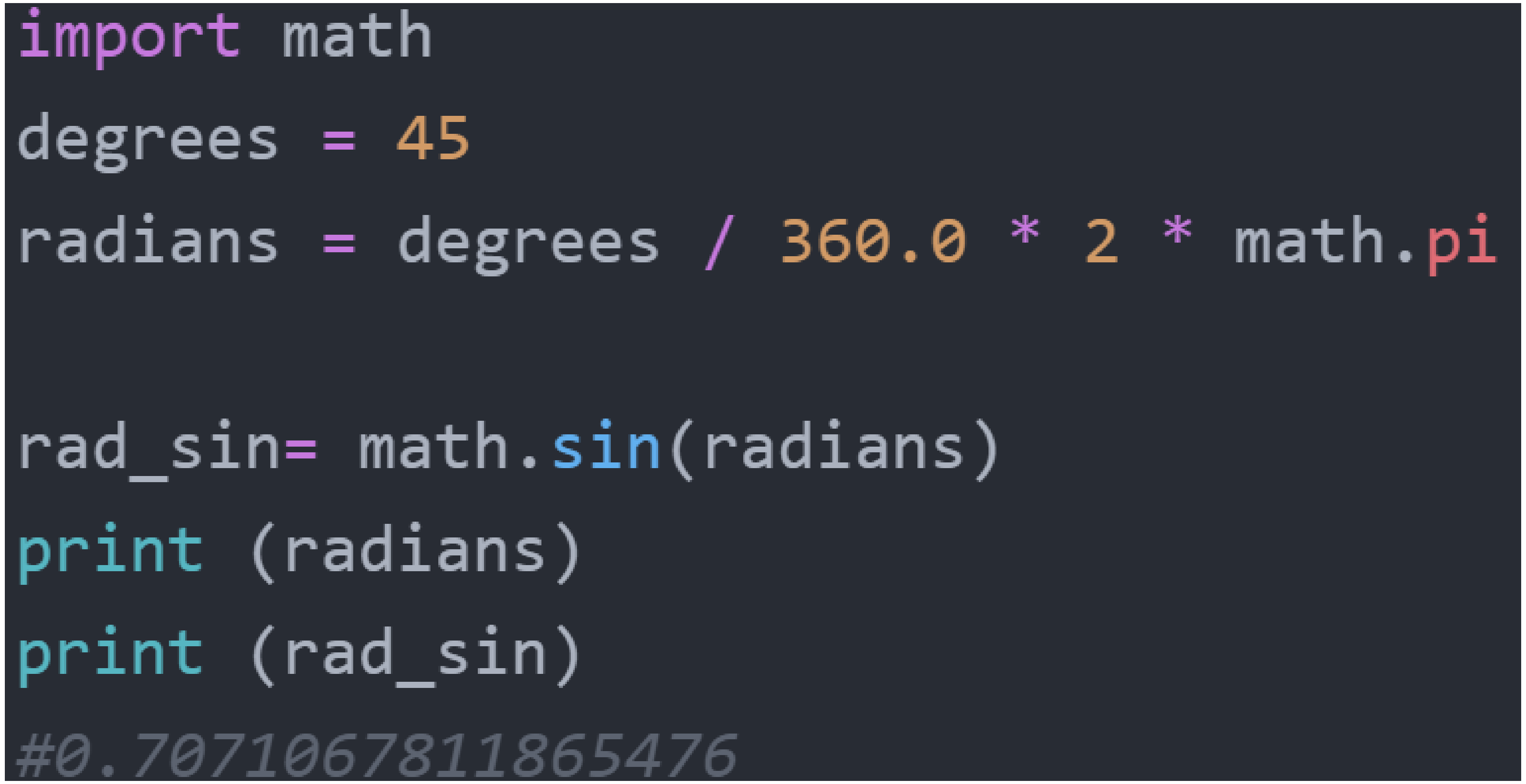
- Python has a math module that provides most of the familiar mathematical functions
- Before we can use the module, we have to import it:

```
>>> import math
```

## Dot Notation

- The module object contains the functions and variables defined in the module
- To access one of the functions, you have to specify the name of the module and the name of the function, separated by a dot (also known as a period)
- This format is called dot notation

## Dot Notation

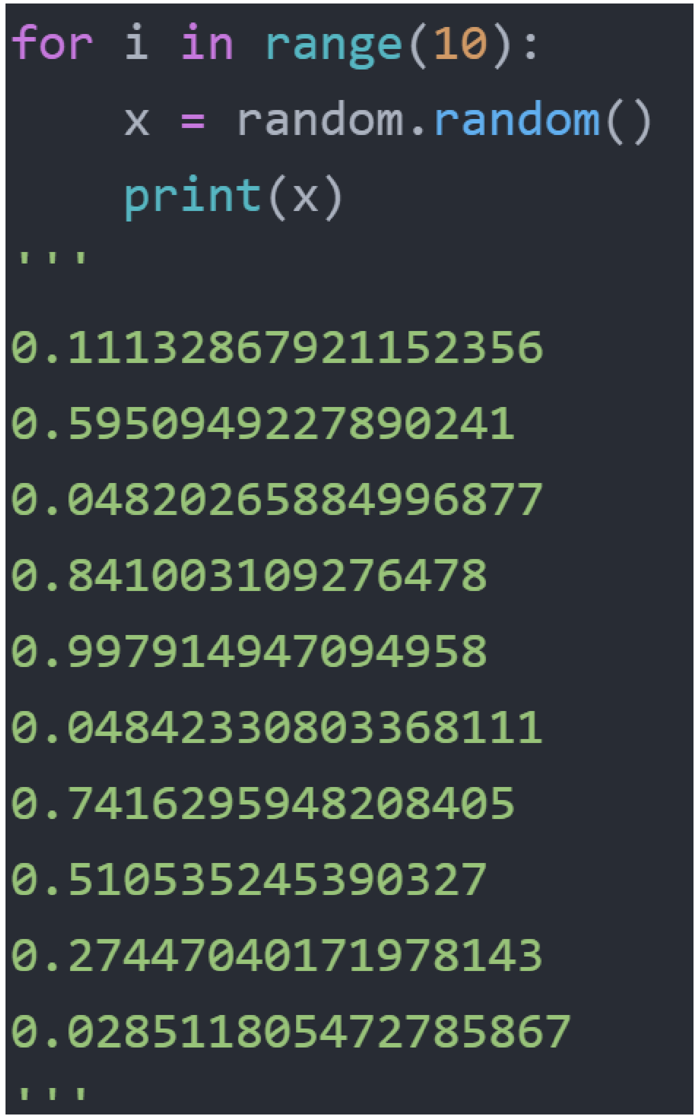


```python
import math
degrees = 45
radians = degrees / 360.0 * 2 * math.pi

rad_sin= math.sin(radians)
print (radians)
print (rad_sin)
#0.7071067811865476
```

## Random numbers

- The random module provides functions that generate pseudorandom numbers



```
for i in range(10):
    x = random.random()
    print(x)
'''
0.11132867921152356
0.5950949227890241
0.04820265884996877
0.841003109276478
0.997914947094958
0.04842330803368111
0.7416295948208405
0.510535245390327
0.27447040171978143
0.028511805472785867
'''
```

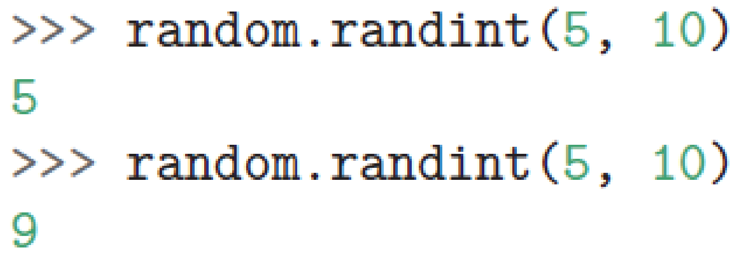
## Activity- 6

- **Run the program on your system and see what numbers you get. Run the program more than once and see what numbers you get.**

## randint()

- The random function is only one of many functions that handle random numbers

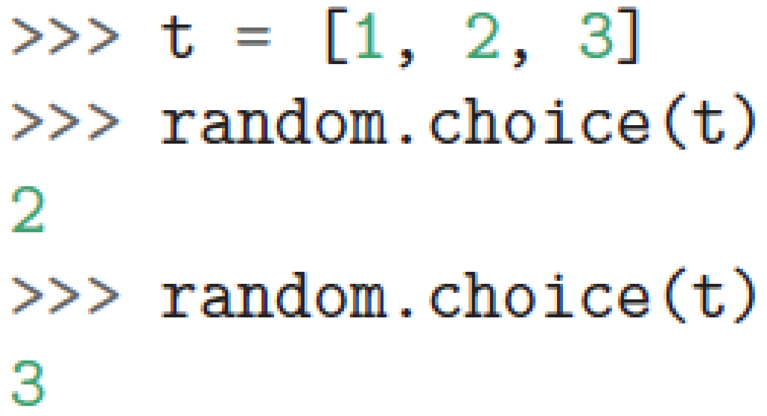
- The function randint takes the parameters low and high, and returns an integer between low and high (including both)



```
>>> random.randint(5, 10)
5
>>> random.randint(5, 10)
9
```

## random.choice()

- To choose an element from a sequence at random, you can use choice:

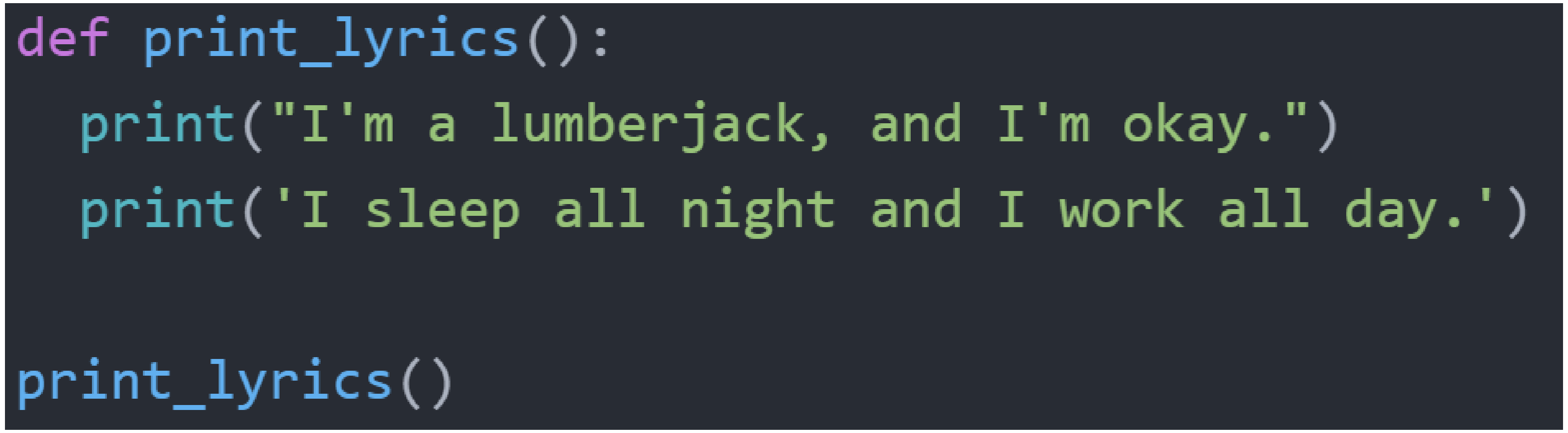


```
>>> t = [1, 2, 3]
>>> random.choice(t)
2
>>> random.choice(t)
3
```

## Adding new functions

- So far, we have only been using the functions that come with Python, but it is also possible to add new functions
- A function definition specifies the name of a new function and the sequence of statements that execute when the function is called
- Once we define a function, we can reuse the function over and over throughout our program

## Adding new functions



```python
def print_lyrics():
  print("I'm a lumberjack, and I'm okay.")
  print('I sleep all night and I work all day.')

print_lyrics()
```

## Rules to create functions

- def is a keyword that indicates that this is a function definition

- The name of the function is print_lyrics

- The rules for function names are the same as for variable names: letters, numbers and some punctuation marks are legal, but thefirst character can’t be a number

- You can’t use a keyword as the name of a function, and you should avoid having a variable and a function with the same name

## Activity- 7

- **Move the last line of this program to the top, so the function call appears before the definitions. Run the program and see what error message you get**

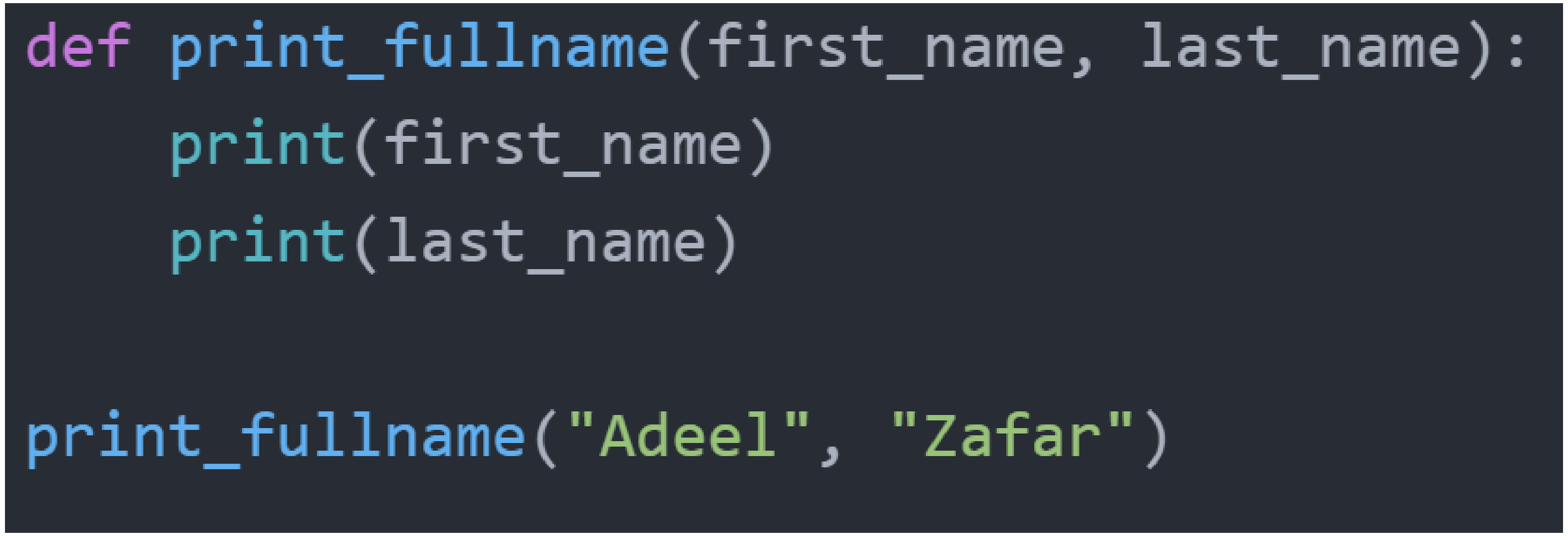
## Flow of execution

- In order to ensure that a function is defined before its first use, you have to know the order in which statements are executed, which is called the flow of execution

## Parameters and arguments

- Some of the built-in functions we have seen require arguments
- For example, when you call math.sin you pass a number as an argument
- Some functions take more than one argument: math.pow takes two, the base and the exponent

## Parameters and arguments



```python
def print_fullname(first_name, last_name):
    print(first_name)
    print(last_name)

print_fullname("Adeel", "Zafar")
```

## Fruitful functions and void functions

- Some of the functions we are using, such as the math functions, yield results; they are fruitful functions

- Other functions, like print_fullname, perform an action but don’t return a value. They are called void functions

## Why functions?

- It may not be clear why it is worth the trouble to divide a program into functions
- There are several reasons:
  - Makes your program easier to read, understand, and debug
  - Functions can make a program smaller by eliminating repetitive code
  - You can reuse them

## Activity - 8

- Write a program to prompt the user for hours and rate per hour using input to compute gross pay. Pay should be the normal rate for hours up to 40 and time-and-a-half for the hourly rate for all hours worked above 40 hours

- Put the logic to do the computation of pay in a function called **computepay()** and use the function to do the computation

- The function should return a value. Use 45 hours and a rate of 10.50 per hour to test the program (the pay should be 498.75). You should use **input** to read a string and **float()** to convert the string to a number. Do not worry about error checking the user input unless you want to - you can assume the user types numbers properly. Do not name your variable sum or use the sum() function.

## Quiz #1

- Rewrite the grade program from the previous slides using a function called **computegrade** that takes a score as its parameter and returns a grade as a string.

| Score | Grade |
|---|---|
| > 0.9 | A |
| > 0.8 | B |
| > 0.7 | C |
| > 0.6 | D |
| <= 0.6 | F |