Crash course on Python (Google)

Jan 18

MAYUR
Date
Page 1

Automation

The process of replacing a manual step with one that happens automatically.

By replacing a manual step with automation that helps us reduce unnecessary manual work.

uses of automation

- The automatic timing and regulation of traffic lights.
- A repetitive task that is at high risk for human error.
- Sending commands to a computer.
- Detecting and removing duplicates of data
- updating a large number of file permissions
- Reporting on system data, like disk or memory usage
- Installing software
- Generating reports

AI involves training a computer machine to perform ~~Aeo tasks without a human needing to progran expli~~ more complex tasks through a process called machine learning.

Function is a reusable block of code that performs a specific task.

Variables are used to temporarily store changeable values in programming code.

Python

=>

Code

```
1 friends = ['Taylor', 'Alex', 'Pat', 'Eli']
2 for friend in friends:
3     print("Hi" + friend)
```

Output

Hi Taylor
Hi Alex
Hi Pat
Hi Eli

* Python interpreter
The program that reads what's in the recipe and ~~tas~~ translates it into instructions for your computer to follow

Common syntax errors

-> Mispellings
-> Incorrect indentations
-> Missing or incorrect key characters:
Bracket types - (curved), [square], {curly}
Quotes types - "straight-double" or 'straight-single'
"curly-double" or "curly-single'
Block introduction characters, like colons -:

→ Data type mismatches
and so on

Common semantic errors:
→ Creating functional code, but getting unintentional outpu
→ Poor logic structures in the design of the code.

Python:

```
For i in range (10):
    print ("Hello, world!")
```

Python is:
→ a general purpose scripting language
→ a popular language used to code a variety of applications.
→ a frequently used tool for automation.
→ a cross-platform compatible language;
→ a beginner friendly language

keywords
→ Reserved words that are used to construct instructions
Values: True, false, None
Conditions: if, elif, else
logical operators: and, or, not
loops: for, in, while, break, continue
Functions: def, return

>> - - -
>> - -

MAYUR
Date
Page

Program

```
1 name = "Brook"
2 print("Hello" + name)          output:
3 #Hello Brook                   Hello Brook
```

```
Print(4+5)          output : 9
```

* Arithmetic operators

x+y	
x-y	
x*y	
x/y	
x**e	Exponent ** operator returns the result of raising x to the power of e
x**2	Square expression returs x squared
x**3	Cube expression returns x cubed
x**(1/2)	Square root (1/2) or (0.5) fractional exponent operator returns the square root of x
x//y	Floor division operator returns the integer part of the integer division of x by y
x%y	Modulo operator returns the remainder part of the integer division of x by y

IDLE (python.py)

Jupyter lab and Jupyter notebook

Order of operations

1> Parentheses ( ), { }, [ ]
2> Exponents x^e (x**e)
3> Multiplication * and Division /
4> Addition + and subtraction -

```
1 ratio = 9 ** 2
2 print (ratio)
```

Code editors and IDEs overview

An IDE is a software application that provides comprehensive facilities for software development.

Colab => google colab | website
abhio.ipynb

Terms and definitions from course 1, module 1

* Client-side scripting language:
  Primarily for web programming; the scripts are transferred from a web server to the end-user internet browser, then executed in the browser
* Code editor
  Tools to provide features, including syntax highlig automatic indentation, error checking and autocomple
* Computer program:
  A step-by-step list of instructions that a compu follows to reach an intended goal.
* Functions: A reusable block of code that performs specific task.
* Interpreter: The program that reads and execute cod
* Platform-specific scripting language: language us by system administrations on those specific platfo

Keywords : in, not, or, for, while, return

operators : +, -, *, /, **, %, //, >, <, ==

Expression: A combination of numbers, symbols, a variables to compute and return a result upon evaluation

Functions: A group of related statements to perform a task and return a value

```
def to-celsius(x):
    '''covert Fahrenheit to celsius'''
    return (x-32) * 5/9


to-celsius(75)
```

Conditional statements: Section of code that direct program execution based on specified conditions

```
number = -4


if number > 0:
    print('Number is positive.')
elif number == 0:
    print('Number is zero.')
else:
    print('Number is negative.')
```

Naming Rules

⇒ Names cannot contain spaces.

⇒ Names may be a mixture of upper and lower case character.

⇒ Names can't start with a number but may con numbers after the first character.

⇒ Variable names and function name should be written in snake_case, which means that all le are lowercase and words are separated using an underscore.

⇒ Descriptive names are better than cryptic abbrevia because they help other programmers (and you) rea and interpret your code.

For eg, Student_name is better than sn. It may feel excessive when you write it, but when you return to your code you'll find it must easier to understand.

⇒
```
print (type("a"))
print(type(2))
print(type(2.5))
```

output

```
<class 'str'>
<class 'int'>
<class 'float'>
```

Eg

```
base = 6
height = 3
area = (base * height)/2
print("The area of triangle is:" + str(area))
```

=> The area of triangle is : 9.0

Implict : It transfer one data type to another automat
cally.

Data type - classes of data (e.g. string, int, float, Boolean etc) which include the properties and behaviors of instances of the data type (variables)

Variable - an instance of data type class, represented by a unique name within the code, that stores changeable values of the specific data type.

~~impli~~ Explicit conversion - when code is written to manually convert one data type to another using a data type conversion function:

str()
int()
float()

* Program 1

```
def greeting(name):      //name is a parameter
    print("welcome," + name)
greeting("Hari")
Greeting("Ram")
```

=> welcome, Hari
welcome, Ram

* Program 2

```
def greeting(name, department):      // two parameter
    print("welcome," + name)
    print("~~rete~~You are a part of " + department)
greeting(Hari, Software Engineering)
greeting(Ram, software Engineering)
```

=> welcome, Hari
You are a part of software engineering

Built-in functions
Built in functions are functions that exist within python and can be called directly. eg print(), type() and str().

⇒ sorted()

The sorted() function sorts the components of a list.

Eg

```
time_list = [12, 2, 32, 19, 57, 22, 14]
print(sorted(time_list))
```

=> [2, 12, 14, 19, 22, 32, 57]

⇒ max() and (min)

```
print(min(time_list))
print(max(time_list))
```

* None

A special data type in python used to indicate that things are empty or that they returned nothing.

Program

```
def area-triangle(base, height):
    return base * height/2
area-a = area-triangle(5,4)
area-b = area-triangle(6,5)
sum = area-a + area-b
print("The sum of both areas is:", + str(sum)
```

* Len is used to calculate length of letters

Eg

```
name = "Kay"
number = len(name) * 9
print ("Hello " + name + ". Your lucky number is"
                                    + str(numb
name = "Cameron"
number = len(name) * 9
print ("Hello" + name + ". Your lucky number is" + str(
```

Output

Hello Kay. Your lucky number is 27.

Hello Cameron. Your lucky number is 63

Comment is done using #

```
def lucky-number(name):
    number = len(name) * 9
    print("Hello" + name + ". Your lucky number is"
                                + str(number))

lucky-number("kay")
lucky-number("Cameron")
```

* Program

```
def circle-area(radius):
    pi = 3.14
    area = pi * (radius**2)
    print(area)

Circle-area(5)
```

output = 78.5

Terms

- Return value - the value or variable returned as the end result of a function.
- parameter (argument) - value passed into a function for use within the function.
- refactoring code - a process to restructure code without changing functionality.

The purpose of the def() keyword is to define a new function.

Conditionals

```
print(10 > 1)
print("cat" == "dog")
print(1 != 2)
```

output
True
False
True

① print ("Yellow" > "cyan" and "Brown" > "Megenta")

⇒ False

② print (25 > 50 or 1 != 2)

⇒ true

③ print (not 42 == "Answer")

⇒ True

④ print (1 < "1")

Error

⑤ print(1 == "1")

False

string comparison is done using unicome

A - 65

a - 97

Print(" ") ⇒ print a blank line

MAYUR
Date
Page

logical operators
They are used to construct more complex expressions.

* Branching with if statements

```
def hint_username(username):
    if len(username) <3:
        print("Invalid username. Must be at least 3
              characters long")
```

⇒ The ability of a program to alter its execution sequence is called branching.

⇒ The body of the if block will only execute when the condition evaluates to true; otherwise its skipped.

* elif statement

```
def hint_username(username):
    if len(username) < 3:
        print("Invalid username. Must be at least 3 characters
                                    long")
    elif len(username) > 15:
        print("Invalid username.")
    else:
        print("valid username")
```

Object oriented Programming

Methods

> Instance methods (for individual object data)
> Class methods (for shared data)
> Static methods (for related tasks)

- Instance methods are the most common type of methods in python.
- Instance methods can take a parameter called self

Class methods are marked with a @classmethod decora
and take a cls parameter that points to the class -
not any specific instance - when the method is called

Static methods, marked with a @staticmethod decorator,
donot take a self or a cls parameter.

* Program

```
Class Apple:
    def __init__(self):
        self.color = "red"
        self.flavor = "sweet"
honeycrisp = Apple()
print(honeycrisp.color)
```

Output

```
red
```

lets override the __str__ method to be more useful for apples:

```
Class Apple:
    def __init__(self, color, flavor):
        self.color = color
        self.flavour = flavour
    def __str__(self):
        return "an apple which is {} and {}".format(
                                self.color, self.flavour)
honeycrisp = Apple("red", "sweet")
print(honeycrisp)
```

output

an apple which is red and sweet

```
Class Triangle:
    def __init__(self, base, height):
        self.base = base
        self.height = height
    def area(self):
        return 0.5 * self.base * self.height
    def __add__(self, other):
        result
        return self.area() + other.area()
triangle1 = Triangle(10,5)
triangle2 = Triangle(6,8)
print("The area of triangle 1 is", triangle1.area(
print("The area of triangle 2 is", triangle2.area(
print("The area of both triangle is", triangle1 + trian
```

output

```
The area ----      is 25.0
"    "       ----  is 24.0
"    "       both --- is 49.0
```

→ Class defines the behavior of all instances of a specific class.

→ The first parameter of the methods, (self), represents current instance.

→ Special methods starts and end with __.

→ It have specific names like __init__ for the constru or __str__ for the conversion to string.

* Examples

```
def product(a, b):
    return(a * b)
print(product(product(2,4), product(3,5))
```

output

120

* Example

```
def get-remainder(x,y):
    if x==0 or y==0 or x==y:
        remainder=0
    else:
        remainder=(x%y)/y
    return remainder
print(get-remainder(70,3))
```

output

0.33333--

While loop

```
x = 0
while x < 5:
    print("Not there yet, x =" + str(x))
    x = x + 1
print("x =" + str(x))
```

output

```
Not there yet, x = 0
Not there yet, x = 1
Not there yet, x = 2
Not there yet, x = 3
Not there yet, x = 4
x = 5
```


While loops instruct your computer to continuou
execute your code based on the value of a
condition.

```
def attempt(n):
    x = 1
    while x <= n:
        print("Attempt" + str(x))
        x += 1
    print("Done")
attempt(5)
```

output

```
Attempt 1
Attempt 2
Attempt 3
Attempt 4
Attempt 5
Done
```

Program

```
x=1
sum=0
while x<10:
    sum=sum+x
    x=x+1
```

output
45 1

```
product=1
while x<10:
    product=product*x
    x=x+1
print(sum, product)
```


* Infinite loop
A loop that keeps executing and never stops

While loop

-> multiplier=1

```
result=multiplier*5
while result<=50:
    if result>50:
        break
    print(result)
    multiplier+=1
    result=multiplier*5
print("Done")
```

output
5
10
15
20
25
30
35
40
45
50
Done

(use while loops when you want to repeat an action until a condition changes)

Common errors in loops

=> Failure to initialize variables.

=> Unintended infinite loops

A pass statement in python is a placeholder statement which is used when the syntax requires a statement but you dont want to execute any code or command

* Program

```
def count_factors(given_number):
    factor = 1
    count = 1
    if given_number == 0:
     return 0
    while factor < given_number:
        if given_number % factor == 0:
            count += 1
            factor += 1
        return count
print(count_factors(0))
print(count_factors(3)
print(count_factors(10))
print(count_factor(24))
```

=>
0
2
4
8

For loops

Eg

```
for x in range(5):
    print(x)
```

0
1
2
3
4

```
Friends = ["Taylor", 'Alex', 'Pat', 'Eli']
for friend in friends:
    print("Hi" + friend)
```

output
Hi Taylor
Hi Alex
Hi Pat
Hi Eli

```
values = [23,52,59,37,48]
sum = 0
length = 0
for value in values:
    sum = sum + value
    length = length + 1
print("Total sum:" + str(sum) + "- Average:" + str(sum/length))
```

Total sum: 219 - Average: 43.8

(use for loops where there's a sequence of elements that you want to iterate

MAYUR Date Page

For loop

- Iterates over a sequence of values

- In Python, and a lot of other programming lang a range of numbers will start with the value by default.
- The list of numbers generated will be one than the given value.

* Program

```
Product = 1
for n in range(1, 10):    // (range from 1 to 9)
    product = product * n
print(product)
```

output : 362880

*
```
def to_celcius(x):
    return (x-32) * 5/9

for x in range(0, 101, 10):
    print(x, to_celcius(x))
```

output

0 - 17.777778

10 - 12.222222

20 - 6.6666667

. - - .

// in the gap of 10 from 0 to 100

0, 10, 20, 30 - -

For loop

→ The (in) keyword, when used with the range() function, generates a sequence of integer numbers, which can be used with a for loop to control the start point, the end point, and the incremental values of the loop.

→ The range() function uses a set of ~~indicates~~ indices that point to integer values, which start at the number 0.

range(start, stop, step)

- Start - Beginning of range
  - value included in range
  - default = 0

- Stop - End of range
  - value excluded from range (to include, use stop+1)
  - no default
  - must provide the index number

- Step - Increment value
  - default value

for loops enable you to interact over a sequence of values such as numbers, names, or lines in a file.

[Nested for loop]

Program

```
for left in range(7):
    for right in range(left, 7):
        print("[" + str(left) + "," + str(right) + "]", end=" ")
    print()
```

output

[0,0] [0,1] [0,2] [0,3] [0,4] [0,5] [0,6]
[1,1] [1,2] [1,3] [1,4] [1,5] [1,6]
[2,2] [2,3] [2,4] [2,5] [2,6]
[3,3] [3,4] [3,5] [3,6]
[4,4] [4,5] [4,6]
[5,5] [5,6]
[6,6]

* teams = ['Dragons', 'Wolves', 'Pandas', 'Unicorns']

```
for home_team in teams:
    for away_team in teams:
        if home_team != away_team:
            print(home_team + "vs" + away_tea
```

output

Dragons vs Wolves
Dragon vs Pandas
Dragon vs Unicorns

String

String represent a sequence of characters and are often used to display output to the user.

* Program

```
greeting = "Hello"
for c in greeting:
    print("The next character is :", c)
```

output

```
The next character is H
 "    "      "     "   e
 "    "      "     "   l
 "    "      "     "   l
 "    "      "     "   o
```

we use for loops with strings to perform tasks and function such as

=> Reading text from file

=> Searching for a value or specific data in document or spreadsheet.

- Looping over a string allows programmers to examine each character within a string individually gather information about its occurence, and perform a given operation.

```
for i in range(len(greeting))
    print(i)
```

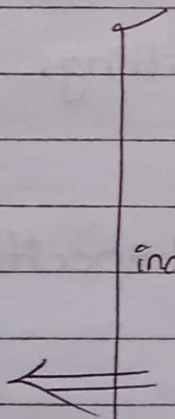
```
0
1
2
3
4
5
```

* Program (while loop)

```
greeting = "Hello"
index = 0
while index < len(greeting):
    print (greeting[index])
    index += 1
```

output

```
H
e
l
l
o
```



while loop with slicing

~~prog~~

```
greeting = "Hello"
index = 0
while index < len(greeting):
    print (greeting[index:index+1])
    index += 1
```

* Program

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x**2 for x in numbers]
print(squared_numbers)
```

=> 1, 4

What is slicing and joining strings?

=> When you slice a string, you extract a subset of the original string - sometimes referred to as indexing a string.

Joining strings is the process of linking two or more strings together to create a bigger string.

* How to slice strings?

```
String1 = "Greetings, Earthlings"
print(string1[0])  # prints "G"
print(string1[4:8]) # prints "ting"
print(string1[11:]) # prints "Earthlings"
print(string1[:5]) # prints "Greet"
```

* Program

```
for x in range(2)
    print("Hello" + str(x))
    for y in range(3+1)
     print("Hi" +str(y)
```

Hell o
Hi
Hi
Hi
Hi
Hello
Hi
))
))
))

* print ("*" *8)

=> ********

~~for a in range(10+1):~~
~~print(a**2)~~

```
for n in range(19);
    if n % 2 == 0
        print(n)
```

(=)

```
for n in range(10)
    print(n+n)
```

* input = "Four score and seven years ago"
for c in input:
    if c.lower() in ['a', 'e', 'i', 'o', 'u'];
    print c

(vowel letter will be printed)

And

print([c for c in input if c.lower() in ['a', 'e', 'i', 'o', 'u']])

* for x in range(1, 11):
    print(x**3)

=> 1
8
27
64
125
1
1

Recursion

the repeated application of the same procedu
to a smaller problem.

Recursion lets us tackle complex problem by
reducing the problem to a simpler one.

In programming, recursion is a way of doing
repetitive task by having a function call itself

```
def factorial(n)
    print("Factorial called with" + str(n))
    if n < 2:
        print("Returning 1")
        return 1
    result = n * factorial(n-1)
    print("Returning" + str(result) + "for factoria
                                    + str(n))
    return result
factorial(4)
```

> Factorial called with 4
" " " 3
" " " 2
" " " 1

Returning 1
Returning 2 for factorial of 2
" 6 " " " 3
" 24 " " " 4

24

* Program

```
def sum_positive_numbers(n):
    if n < 1:
        return 0
    return n + sum_positive_numbers(n-1)
print(sum_positive_numbers(3))
print(sum_positive_numbers(5))
```

=> 6
15

[To override the insertion of the new line character and replace it with a space, add end = " " as the last item in the print() parameters.
[i.e print(x+1, end " ")]]

* For loop with the range() function

```
def count_by_10(end)
    count = ""
    for number in range(0, end+1, 10):
        count += str(number) + " "
    return count.strip()
print (count_by_10(100))
```

=> 0 10 20 30 40 50 60 70 80 90 100

* Nested for loop with range() function for ma

```
def matrix(initial_number, end_of_first_row):
    n1 = initial_number
    n2 = end_of_first_row+1
    for column in range(n1,n2)
        for row in range (n1, n2)
            print (column * row, end=" ")
        print()
matrix(1,4)
```

=>
```
1  2  3  4
2  4  6  8
3  6  9  12
4  8  12 16
```


* Nested for loop with range() function

```
for outer_loop in range(10):
    for inner_loop in range (outer_loop):
        print (inner loop
```

=>
```
0
0
1
0
1
2
0
1
2
3
0
1
2
```

MAYUR
Date
Page

with end = " "
prints in — — — — order

without end = " "
prints in
order

* use a while loop to print a sequence of numbers.

```
Starting-number = 18
while starting-number >=0:
    print (starting-number, end = " ")
    staring-number -= 3
```

=> ~~18,15,12,9~~ 18 15 12 9 6 3 0

* use a while loop to count the number of digits in a numerical value.

```
def x-figure(salary):
    tally = 0
    if salary == 0:
        tally += 1
    while salary >= 1:
        salary = salary/10
        tally += 1
    return tally
print ("The CEO has a " + str (x-figure(2300000)) + "-figure salary.")
```

=> The CEO has a 7-figure salary

* use nested if-else statements and while loops to count up or count down from the first variable to the second variable.

```
def elevator-floor(enter, exit):
    eter
    floor = enter
    elevator_direction = " "
    if enter > exit
        elevator_direction = "Going down:"
        while floor >= exit:
            elevator-direction += str(floor)
            if floor > exit:
                elevator-direction += "1"
            floor -= 1
    else:
        elevator_direction = "Going up:"
        while floor <= exit:
            elevator-direction += str(floor)
            if floor < exit:
                elevator-direction = "1"
            floor += 1
    return elevator_direction
print (elevator-floor(1,4))
print (elevator-floor(6,2)
```

⇒ Going up: 1121314
Going down: 6151413121

String

String is a datatype in python that is used to re
a piece of text.

* name = "Jaylen"
print(name[1]) => a

name = "Jaylen"
print(name[0]) => J

* text = "Random string with alot of characters"
print (text[-1])
print (text[-2])

=> s
r

* Color = "orange"
Color[1:4]
=> ran

Slice

→ The portion of a string that can contain mor
than one character; also sometimes called a
substring.

* program

```
pets = "Cats & Dogs"
pets.index("s")
```

=> 3

* Program

```
pets = "Cats & Dogs"
"Dragons" in pets   => True
"Cats" in pets      => False
```

* Upper and Lower case

```
"Mountains".upper()   => MOUNTAINS
"Mountains".lower()   => mountains
```

* "yes".strip() => yes (removes space both side)
"yes".rstrip() => yes (removes space right)
"yes".lstrip() => yes (removes space left)

* "what".count("h") => 1

* "Forest".endwith("rest") => Yes

with {:.2f} => 7.50 => In .format
without { } => 7.5
{:>3} => three place greater

MAYUR
Date
Page

* "Forest".isnumeric() => False
"12345".isnumeric() => True

* int("12345") + int("54327") => 66666

* "..".join(["This", "is", "a", "phrase", "joined", "by",

=> This.. is.. a.. phrase.. joined ---

* "This is another example".split()
=> ['This', 'is', 'another', 'example']

* name = "Manny"
number = len(name) * 3
print("Hello {}, your lucky number is {}".format
(name, number))

=> Hello Manny, your lucky number is 15

* print(test.replace("wood", "plastic"))

=> Then wood will be replaced by plastic

Formating expressions

Expr	Meaning	Example
{:d}	integer value	"{0:.f}".format(10.5) -> '10'
{:.2f}	floating point with that many decimals.	"{:.2f}".format(0.5) -> '0.50'
{:.2s}	string with that many characters	'{:.2s}'.format('Python') -> 'Py'
{:<6s}	String aligned to the left that many spaces.	'{:<6s}'.format('Py') -> 'Py '
{:>6s}	String aligned to the right that many spaces.	'{:>6s}'.format('Py') -> ' Py'
{:^6s}	String centered in that many spaces	'{:^6s}'.format('Py') -> ' Py '

=> Use for loop to iterate through each letter of a string.

```
def mirrored-string(my-string):
    forwards = " "
    backwards = " "
    for character in my-string:
        if character.isalpha():
            forwards += character
            backwards = character + backwards
    if forwards.lower() = backwards.lower():
        return True
    return False
print(mirrored-string("12 Noon"))
print(mirrord-string("was it a car or cat I saw"))
print(mirrord-string("eve, Madam eve"))
```

=>
True
False
True

* format()

```
def convert_weight(ounces):
    pounds = ounces/16
    result = "{} ounces equals {:.2f} pounds".format(ounces, pounds)

    return result
print(convert_weight(12))
print(convert_weight(50.5))
print(convert_weight(16))
```

=> 12 ounces equals 0.75 pounds
=> 50.5 ounces equals 3.16 pounds
=> 16 ounces equals 1.00 pounds

* format() parameters

```
def username(last_name, birth_year):
    return {"{}{}".format(last_name[0:3], birth
print(username("Ivanov", "1985"))
print(username("Rodriguez", "2000"))
print(username("Deng", "1997"))
```

=> Iva1985
Rod2000
Den1997

Replace() and len()

```
def replace-date(schedule, old-date, new-date):
    if schedule.endswith(old-date):
        p = len(old-date)
        new-schedule = schedule[:-p] + schedule[-p:].replace
                                        (old-date, new-date)
        return new-schedule
    return schedule
print (replace-date("last year's annual report will be
    released in march 2023", "2023", "2024"))
```

=>

last year's annual report will be released on march 2024.

List

```
x = ["Now", "we", "are", "cooking!"]
type(x)
```

=> <class 'list'>

```
fruits = ["Pineapple", "Banana", "Apple", "Melon"]
fruits.inserts(0, "orange")
fruits.inserts(25, "Peach")
fruits.remove("Melon")
print(fruits)
```

=> ["orange", "Pineapple", "Banana", "Apple", "Peach"]

```
fruits.pop(3)
print(fruits)
```

=> ["orange", "Pineapple", "Banana", "Peach"]

```
fruits[2] = "strawberry"
print(fruits)
```

["orange", "Pineapple", "Strawberry", "Peach"]

```
fruits.append("Kiwi")
print(fruits)
```

['orange', 'Pineapple', —, —, —, 'Kiwi']

Lists and Tuple

*

```
def convert_seconds(seconds):
    hours = seconds // 3600
    minutes = (seconds - hours * 3600) // 60
    remaining_seconds = seconds - hours * 3600 - minutes * 60
    return hours, minutes, remaining_seconds
results = convert_seconds(5000)
type(result)
print(results)
```

=> <class 'tuple'>
(1, 23, 20)

Strings => Sequence of characters, and are immutable
Lists => Sequence of elements of any type and are mutable.
Tuples => Sequence of elements of any type, that are immutable
The position of the elements inside the tuple have meaning.

*

```
animals = ["Lion", "Zebra", "Dolphin", "Monkey"]
chars = 0
for animal in animals:
    chars += len(animal)
print("Total characters : {}, Average length: {}".format
                    (chars, chars/len(anima)))
```

=> Total character : 22, Average length : 5.5

* winners = ["Ram", "Hari", "Gita"]
for index, person in enumerate(winners):
    print("{} - {}".format(index+1, person)

=> 1 - Ram
2 - Hari
3 - Gita

* def full-name(people)
    result = []
    for email, ~~person~~ name in people:
        result.append("{} <{}>".format(name, email
    return result
    print(full-name([["alex@example.com", "Alex Die

=> ['Alex Diego <alex@example.com>']

* multiples = []
for x in range(7, 77):
    multiples.append(x*7)
print(multiples)

=> [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]

* languages = ["Python", "Perl", "Ruby", "Go", "Java"]
lengths = [len(language) ~~for~~ for language in languages]
print(lengths)

=> [6, 4, 4, 2, 4]

* z = [x for x in range(0, 101) if x%3 == 0]
print(z)

=> [0, 3, 6, 9, 12, 15, 18 ---- 99]

* For loop and list comprehension

```
print ("List comprehension result:")
print([x*2 for x in range(1, 11)])
print ("long form code result:")
my-list = []
for x in range (1, 11):
    my-list.append(x*2)
print(my-list)
```

=> list comprehension result:
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
long form code result:
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

List comprehension with conditional statements

```
print("List comprehension result:")
print([x for x in range(1, 101) if x%10==0])
print("long form code result:")
my-list[]
for x in range(1, 101):
    if x%10 == 0:
        my-list.append(x)
print(my-list)
```

List comprehension result:
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
long form code result:
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

List-specific operations and methods

List[index] = x ⇒ Replaces the element at index
with x.

List.append(x) ⇒ Appends x to the end of the li

List.insert(index, x) ⇒ Inserts x at index positi
[inde

List.pop(index) - Returns the element at [index
and removes it from the list. if [index] posi
is not in the list, the last element in the l
is returned and removed.

List.remove(x) - Removes the first occurence o
x in the list.

- list.sort() - Sorts the items in the list.
- list.reverse() - Reverses the order of items of the list.
- list.clear() - Deletes all items in the list.
- list.copy() - Creates a copy of the list.
- list.extend(other-list) - Appends all the elements of other-list at the end of list.
- map(function, iterable) - Applies a given function to each item of an iterable (such as a list) and returns a map object with the results
- Zip(*iterables) - Takes in iterables as arguments and returns an iterator that generates tuples, where the i-th tuple contains the i-th element from each of the argument iterables.

Tuple use cases
=> Protecting data
=> Hashable keys
=> Efficiency

* Tuple() operator

```
my-list = [1, 2, 3, 4]
my-tuple = tuple(my-list)
print(my-tuple)
```

=> (1, 2, 3, 4)

Tuples with mutable objects

```
my-tuple = (1, 2, ['a', 'b', 'c'])
my-tuple[2][0] = 'x'
print(my-tuple)
```

> (1, 2, ['x', 'b', 'c'])

Returning multiple values from functions

```
def calculate-numbers(a,b):
    return a+b, a-b, a*b, a/b
result = calculate-numbers(10,2)
print(result)
```

=> (12, 8, 20, 5.0)

Skill

Use a for loop to modify elements of a li

```
years = ["January 2023", "May 2025", "April 2023"
    "August 2024", "September 2025", "Decemb
updated-years = []
for year in years:
    if year.endswith("2023")
        new = year.replace("2023", "2024")
        updated-years.append(new)
else:
    updated-years.append(year)
print(updated-years)
```

=> ["January 2024", "May 2025", "April 2024", "August 2024", "September 2025", "December 2024"]

* Use a list comprehension to modify elements of a list.
Use the string.replace() method within a list comprehension.
Use the string[index] method within a list comprehension.

```
years = ["January 2023", "May 2025", "April 2023",
         "August 2024", "September 2025", "December 2023"]
updated-years = [year.replace("2023", "2024") if year[-4:]
                 == "2023" else year for year in years]
print(updated-years)
```

=> ["January 2024", "May 2025", "April 2024", "August 2024", "September 2025", "December 2024"]

* Use the ~~split~~ string.split() method to separate a string into a list of individual words.

```
def change-string(given_string):
    new-string = ""
    new-list = given-string.split()
    for element in new-list:
        new-string += element[1:] + "-" + element[0]
                                  + " "
```

```
return new_string
print(chang_string("1one 2two 3three 4four 5fi
```

=> one-1 two-3 four-4 five-5

* use the string.join() method to concatenate a string that provides a list name and its element

```
def list_elements(list_name, elements):
    return "The" + list_name + "list includes:" + ", "
                                .join(elements)
print(list_elements("Printers", ["color printer", "Black
    white Printer", "3-D printer"])
```

=> The Printers list includes : color Printer, Black ar white Printer, 3-D Printer

* use map() and convert the map object to a lis so we can print all the results at once.

```
def add_one(number):
    return number+1
numbers = [1,2,3,4,5]
results = map(add_one, numbers)
print(list(result))
```

=> [2,3,4,5,6]

* Use zip() to combine a list of names and ages into a list of tuples, and print all the tuples at once.

```
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 35]
combined = zip(names, ages)
print(list(combined))
```

=> [('Alice', 25), ('Bob', 30), ('Charlie', 35)]

* ```
def biography-list(people):
    for person in people:
        name, age, profession = person
        print("{} is {} years old and works as a {}."
              .format(name, age, profession))
biography-list([("Ira", 30, "a chef"), ("Raj", 35, "a lawyer")])
```

=> Ira is 30 years old and works as a chef.
Raj is 35 " " " " " " lawyer.

## Dictionaries

```
x = {}
type(x)
```

=> <class 'dict'>
```

* file-counts = {"jpg": 10, "txt": 14, "csv": 2, "py": 23}
file-counts["txt"]
=> 14

* file-counts = {"jpg": 10, "txt": 14, "csv": 2, "py": 23}
file-counts["cfg"] = 8
print(file-counts)

=> {'jpg': 10, 'txt': 14, 'csv': 2, 'py': 23, 'cfg': 8}

del file-counts["cfg"]

-> The data inside dictionaries take the form of pairs of keys and values.
-> Dictionary are mutable (can be changed and removed)

* file-counts = {"jpg": 10, "txt": 14, "csv": 2, "py": 2
for exct, amoont in file-counts.items():
print("There are {} files with the .{} extens
.format(amount, ex

=> There are 10 files with the .jpg extension
There are 14 files with the .txt "
" " 2 " " " .csv "
" " 23 " " " .py "

MAYUR
Date
Page

* file-counts = {"jpg": 10, "txt": 14, "csv": 2, "py": 23}
file-counts.keys()
file-counts.values()

=> dict-keys(['jpg', 'txt', 'csv', 'py'])
dict-values([10, 14, 2, 23])

* file-counts = {"jpg": 10, "txt": 14, "csv": 2 "py": 23}
for value in file-counts.values():
    print(value)

=> 10
14
2
23

* def count-letters(text):
    result = {}
    for letter in text:
        if letter not in result:
            result[letter] = 0
        result[letter] += 1
    return result
count-letters("aaaa")
count-letters("a long string with a lot of letters")

=> {'a': 4}
{'a': 2, ' ': 7, 'l': 3, 'o': 3, 'n': 2, 'g': 2, 's': 2, 't': 5

Set

=> used when you want to store a bunch of elem
and be certain that they're only present once.

Operations

- len(dictionary) - Returns the number of items in dictionary
- for key, in dictionary - Iterates over each key, ~~val pair~~ in a dictionary.
- for key, value in dictionary.items() - Iterates ov each key, value pair in a dictionary.
- if key in dictionary - checks whether a key is a dictionary.
- dictionary[key] - Accesses a value using the associ key from a dictionary.
- dictionary[key] = value > sets a value associated a key.
- del dictionary[key] > Removes a value using th associated key from a dictionary.

Methods

- dictionary.get(key, default) -> Returns the value corresponding to a key, or the default valu if the specified key is not present.
- dictionary.keys() > Returns a sequence contai the keys in a dictionary.
- dictionary.values() > Returns a sequence contain the values in dictionary.

dictionary[key].append(value) => Appends a new value for an existing key.
dictionary.update(other-dictionary) => updates a dictionary with the items from another dictionary. Existing entries are updated; new entries are added.
dictionary.clear() => Deletes all items from a dictionary
dictionary.copy() => Makes a copy of a dictionary

Both dictionaries and Lists:
are used to organize elements into collections;
are used to initialize a new dictionary or list, use empty brackets;
can interate through the items or elements in the collection
can use a variety of methods and operations to create and change the collections, like removing and inserting items or elements.

Dictionaries only:
are unordered sets;
have keys that can be a variety of data types, including strings, integers, floats, tuples;
can access dictionary values by keys;
use square brackets inside curly brackets {[]};
use colons between the keys and the values(s);
use commas to separate each key group and each value within a key group;

- Make it quicker and easier for a python interpreter to find specific elements, as compar to a list

Lists only:
- are ordered sets;
- access list elements by index positions;
- require that these indices be integers;
- use square brackets [];
- use commas to separate each list element.

* Using a for loop with the dictionary.items() method to calculate the sum of the values in a dictionary.

```
def sum-server-use-time(server):
    total-use-time = 0.0
    for key,value in server.items():
        total-use-time += server[key]
    return round(total-use-time,2)
fileserver = {"EndUser1":2.25, "EndUser2":4.5,
              "EndUser3":1, "EndUser4":3.75,
              "EndUser5":0.6, "EndUser6":8}
print(Sum-server-use-time(Fileserver))
```

=> 20.1

using the list.append(x) method
using nested for loops with the dictionary.items()

```
def list_full_names(employee_dictionary):
full_names = []
for last_name, first_names in employee_dictionary.items():
    for first_name in first_names:
        full_name.append(first_name + " " + last_name)
return(full_names)
print(list_full_names({"Ali": ["Muhammad", "Amir", "Malik"],
                       "Devi": ["Ram", "Amaira"],
                       "Chen": ["Feng", "Li"]}))
```

['Muhammad Ali', 'Amir Ali', 'Malik Ali', 'Ram Devi',
'Feng Chen', 'Li Chen']

dictionary[key] = value
Use nested for loops and an if-statement, and the dictionary.items() method.
Use the dictionary[key].append(value)

```
def invert_resource_dict(resource_dictionary):
    new_dictionary = {}
    for resource_group, resources in resource_dictionary
                                        .items():
        for resource in resources:
            if resource in new_dictionary:
```

```
            new-dictionary[resource].append(resource-gro
        else:
            new-dictionary[resource] = [resource-group
    return(new-dictionary)
print(~~int~~ invert-resource-dict({"Hard Drives": ["IDE H
    "SCSI HDDs"], "PC Parts": ["IDE HDDs", "SCSI HDD
    "High-end video cards", "Basic video cards"]})
```

=> {'IDE HDDs': ['Hard Drives', 'PC Parts'], 'SCSI HDDs': ['Hard Drives', 'PC Parts'], 'High-end video cards': ['PC Parts', 'video cards'], 'Basic video cards': ['PC Part 'video cards']}

Terms

* Dictionaries: A data type used to organize elements into collections, taking the form of pairs of keys and values.
* list comprehensions: Create new List based on sequences or ranges
* String: A data type used to represent a piece of text. sequences of characters and are immutable
* Tuples: Sequences of elements of any type that are immutable, written parentheses instead of square brackets.

Operations, methods, and functions

- string methods, operations, and functions
  - .upper()
  - .lower()
  - .split()
  - .format()
  - .isnumeric()
  - .isalpha()
  - .replace()
  - string.index[]
  - len()

- list operations and methods
  - .reverse()
  - .extend()
  - .insert()
  - .append()
  - .remove()
  - .sort()
  - list comprehensions[]
  - list comprehensions[] with if condition

Dictionary operations and methods

- .items()
- .update()
- .keys()
- .values()
- .copy()
- dictionary[key]
- dictionary[key] = value

Using string method

```
def sales-prices(item-and-price):
    item = ""
    price = ""
    item-or-price = item-and-price.split()
    for x in item-or-price:
        if x.isalpha():
            item += x + " "
        else:
            price = x
    item = item.strip()
    return "{} are on sales for ${}".format(item,pric
print(sales-prices("Winter fleece jacket 49.99"))
```

=> Winter fleece jackets are on sale for $49.99

- Use the len() function to measure a string.

```
def coont-words(data-field):
    split-data = data-field.split()
    return len(split-data)
coont-words("Catalog item 3523: organic raw
                pumking seeds in shell")
```

=> 9

Skill 2: Using list methods
- reverse()
- extend()

Combine

```
def record-profit-years (recent-first, recent-last):
    recent-first.reverse()
    recent-last.extend (recent-first)
    return recent-last
recent-first = [2022, 2018, 2011, 2006]
recent-last = [1989, 1992, 1997, 2001]
print (record-profit-years (recent-first, recent-last))
```

=> [1989, 1992, 1997, 2001, 2006, 2011, 2018, 2022]

Skill 3: Using a list Comprehension
include a calculation with a for loop in a range with 2 parameters (lower, upper+1).

```
def list-years(start, end):
    return [year for year in range(start, end+1)]
print(list-years(1972, 1975))
```

=> [1972, 1973, 1974, 1975]

- use a list comprehension [] with a for loop and a
if condition.

```
def odd-numbers(x,y):
    return [n for n in range(x,y) if n%2!=0]
print(odd-number(5,15))
```

-> [5, 7, 9, 11, 13]

Skill 4: Using dictionary methods
- Iterate through the keys and values of a dictio
- Return the keys and values in a formatted string using .format() function

```
def network(servers):
    result = " "
    for hostname, IP-address in servers.items():
        result += "The IP address of the {} server i
                  {}".format(hostname, IP-address)+"
    return result
print(network({"Domain Name Server": "8.8.8.8",
               "Gateway Server": "192.168.1.1"})
```

=> The IP address of the Domain Name server is 8.8.8.8
The IP address of the Gateway Server is 192.168.1.1

```
return sentence.replace(word, word.upper())
```

```
def reset-scores (game-scores):
    new-game-scores = game-scores.copy()
    for player, score in new-game-scores.items():
        new-game-scores[player] = 0
    return new-game-scores
game1-scores = {"Arshi": 3, "Catalina": 7, "Diego": 6}
print (~~reset~~ reset-scores(game1-scores))
```

=> {'Arshi': 0, 'Catalina': 0, 'Diego': 0}

* Event class
Contains the date when the event happened, the name of the machine where it happened, the user involved, and the event type

```
names = ["Carlos", "Rag", "Alex", "Kelly"]
print (sorted (names))
print (names)
print (sorted (names, key = len))
```

=> ['Alex', 'Carlos', 'Kelly', 'Rag']
['Carlos', 'Ray', 'Alex', 'Kelly']
['Rag', 'Alex', 'Kelly', 'Carlos']

```
def get_event_date(event):
    return event.date

def current_users(events):
    events.sort(key=get_event_date)
    machines ={}
    for event in events:
        if event.machine not in machines;
            machines[event.machine] =set()
        if event.type == "login":
            machines[event.machine].add(event.use
        elif event.type == "logout":
            machines[event.machine].remove(event.
    return machines
def generate_report(machines):
    for machine, users in machines.items():
        if len(users)>0:
            user_list = ",".join(users)
            print("{}:{}".format(machine, user_lis
class Event:
    def __init__(self, event_date, event_type, machine
                 name, user):
        self.date = event_date
        self.type = event_type
        self.machine = machine_name
        self.user = user
events = [
```

```
Event('2020-01-21 12:45:46', 'login', 'myworkstation
                                .local', 'jordan'),
Event('2020-01-22 15:53:42', 'logout', 'webserver
                                .local', 'jordan'),
]
users = current_users(events)
print(users)
```

=> {'webserver.local': {'lane'}

webserver.local: lane