1. msg = 'Hello Lloyd you are ' + str(21)
2. print(msg)

**Converting Other Type into Strings**

1. some\_string = 'Hello World'
2. print('Testing a String')
3. print('-' \* 20)
4. print('some\_string', some\_string)
5. print("some\_string.startswith('H')",
6. some\_string.startswith('H'))
7. print("some\_string.startswith('h')",
8. some\_string.startswith('h'))
9. print("some\_string.endswith('d')", some\_string.endswith('d'))
10. print('some\_string.istitle()', some\_string.istitle())
11. print('some\_string.isupper()', some\_string.isupper())
12. print('some\_string.islower()', some\_string.islower())
13. print('some\_string.isalpha()', some\_string.isalpha())
14. print('String conversions')
15. print('-' \* 20)
16. print('some\_string.upper()', some\_string.upper())
17. print('some\_string.lower()', some\_string.lower())
18. print('some\_string.title()', some\_string.title())
19. print('some\_string.swapcase()', some\_string.swapcase())
20. print('String leading, trailing spaces', " xyz ".strip())

**Other Strings Operations**

>> -1

1. print('Edward John Rawlings'.find('Alun'))

>> 5

1. print('Edward Alun Rawlings'.find('Alun'))

**Finding Sub Strings**

String.find(string\_to\_find)

The method returns −1 if the string is not present. Otherwise it returns an index indicating the start of the substring

**Replacing Strings**

>> Goodbye World!

1. welcome\_message = 'Hello World!'
2. print(welcome\_message.replace("Hello", "Goodbye"))

>> my\_string.count(' '): 8

1. my\_string = 'Count, the number of spaces'
2. print("my\_string.count(' '):", my\_string.count(' '))

**Counting Strings**

1. title = 'The Good, The Bad, and the Ugly'
2. print('Source string:', title)
3. print('Split using a space')
4. print(title.split(' '))
5. print('Split using a comma')
6. print(title.split(','))

**Python variables**

*Python as Dynamic Typing. That is the type of the data held by a variable can Dynamically change as the program executes*

>>Source string: The Good, The Bad, and the Ugly

>>Split using a space

>>['The', 'Good,', 'The', 'Bad,', 'and', 'the', 'Ugly']

>>Split using a comma

>>['The Good', ' The Bad', ' and the Ugly']

**Splitting Strings**

1. print (‘\*’ \* 10)
2. print (‘Hi’ \* 10)

**Repeating Strings**

**Prompting for numerical input**

1. High= input (“enter your body high: “)
2. High = int (High)
4. Pi= input (“enter pi value : “)
5. Pi = float (Pi)

1. User\_name = input (“Enter your name :”)
2. Print (“hello”, “ “, User\_name)

**Prompting a value**

**input () function**

*to use when it prompts user for input. All input will stored as a string data.*

1. A = “hello world”
2. Print(A)

 >> hello world

**Hello world with a variable**

Print (“Hello World”)

**Hello world**

>> \*\*\*\*\*\*\*\*\*\*

>> HiHiHiHiHiHiHiHiHiHi

1. my\_string = 'Hello World'
2. print(my\_string[4]) # characters at position 4
3. print(my\_string[1:5]) # from position 1 to 5
4. print(my\_string[:5]) # from start to position 5
5. print(my\_string[2:]) # from position 2 to the end

**Accessing Character form a String**

>> Good day

1. String\_1=’Good’
2. String\_2=”day”
3. String\_3= String\_2 + String\_2
4. print(string\_3)

**String Concatenation**

1. ‘Hello World’
2. “Hello World”

1. my\_variable = 'John'
2. print(my\_variable)
3. my\_variable = 42
4. print(my\_variable)
5. my\_variable = True
6. print(my\_variable)

**Representing Strings**

**Python Strings**

*a string is a series, or sequence, of characters in order. In this definition a character is anything you can type on the keyboard in one keystroke*

**Beginner’s Guide to Python 3 Programming**

**print () function**

*it will print whatever user give, when user give a string it will print string, when it’s given an integer it’s will print integer, also for other data type as float and others*

**Python as interpreted language**

*An interpreted language is one that does not require a separate compilation phase to convert the human readable format into something that can be executed by a computer. Instead the plain text version is fed into another program (generally referred to as the interpreter) which then executes the program for you.*

**Python Programming Paradigms**

*Python is hybrid programming language as it allows you to write very procedural code, to use objects in an object oriented manner and to write functional programs*

1. **Procedural Programming**: represented as a sequence of instructions that tell the computer what it should do explicitly.
2. **Declarative Programming**: allow developers to describe how a problem should be solved, with the language/environment determining how the solution should be implemented.
3. **Object Oriented Programming**: approaches that represent a system in term of the objects that system. Each object ca hold its own data (also known as state) as well as define behavior that defines what object can do.
4. **Functional programming**: Language decompose a problem into a set of functions. Each function is independent of any external state, operating only on the inputs they received to generate their outputs.

>>John

>>42

>>True

Testing a String

--------------------

some\_string Hello World

some\_string.startswith('H') True

some\_string.startswith('h') False

some\_string.endswith('d') True

some\_string.istitle() True

some\_string.isupper() False

some\_string.islower() False

some\_string.isalpha() False

String conversions

--------------------

some\_string.upper() HELLO WORLD

some\_string.lower() hello world

some\_string.title() Hello World

some\_string.swapcase() hELLO wORLD

String leading, trailing spaces xyz

1. number=int(input("Enter a number:"))
2. faktorial=1
3. if number<0:
4. print("The factorial is not defined")
5. elif number==0:
6. print("0! =", faktorial)
7. else:
8. for i in range(1,number+1):
9. faktorial=faktorial\*i
10. print(number,"! = ",faktorial)

**Factorial using for loop**

1. winner = None
2. print('winner:', winner)
3. print('winner is None:', winner is None)
4. print('winner is not None:', winner is not None)

1. print('Only print code if all iterations completed')
2. num=int(input('Enter a number to check for:'))
3. for i in range(0,6):
4. if i==num:
5. break
6. print(i, ' ', end='')
7. else:
8. print()
9. print('All iterations succesful')

**For loop with else**

The else part is executed if and only if all items in the sequence are processed

1. for i in range (0,10):
2. if i % 2==1:
3. continue
4. print(i, ' ', end='')
5. print()
6. print('Done')

*For in range*

1. print('Print out values in a range')
3. for i in range(0,10):
4. print(i, ' ', end='')
5. print()
6. print("Done")

*For in range with increment*

1. print('Print out values in a range')
3. for i in range(0,10):
4. print(i, ' ', end='')
5. print()
6. print("Done")

*Print out something in for*

1. for \_ in range(0,10):
2. print('\*', end='')
3. print()

**Break loop**

1. print('only print code if all iterations completed')
2. num= int(input('Enter a number to check for:'))
3. for i in range(0,6):
4. if i==num:
5. break
6. print(i,' ',end='')
7. print('Done')

**Continue loop**

**For Loop**

for i = from 0 to 10

statement or statements

for <variable-name> in range(...):

statement statement

1. count=0
2. Upper\_limit=int(input("Enter the upper limit:"))
3. print('Starting')
4. while count<=Upper\_limit:
5. print(count, ' ', end='')
6. count+=1
7. print()
8. print('Done')

**While Loop**

while <test-condition-is-true>:

statement or statements

**If expression**

<result1> if <condition-is-met> else <result2>

**Iteration/Looping**

*The while loop and the for loop available in Python. These loops are used to control the repeated execution of selected statements.*

1. age=int(input("enter your age:"))
2. Status=None
4. Status=('teenager' if age>12 and age<20 else 'note teenager')
5. print(Status)

1. savings = float(input("Enter how much you have in savings: "))
2. if savings == 0:
3. print("Sorry no savings")
4. elif savings < 500:
5. print('Well done')
6. elif savings < 1000:
7. print('Thats a tidy sum')
8. elif savings < 10000:
9. print('Welcome Sir!')
10. else:
11. print('Thank you')

**The Use of elif**

1. num = int(input('Enter yet another number: '))
2. if num < 0:
3. print('Its negative')
4. else:
5. print('Its not negative')

**Using else in an If**

1. num = int(input('Enter a number: '))
2. if num < 0:
3. print(num, 'is negative')

**Using If**

and -> Returns True if both left and right are true

or -> Returns two if either the left or the right is truce

not -> Returns true if the value being tested is False

**Logical Operator**

= -> equal

!= -> not equal

< -> less than

> -> greater than

<= -> less than or equal

>= -> greater than or equal

1. X=1 #integer
2. X=1.0 #float
3. X = 2 + 2 j #complex

**Comparison Operator**

if <condition-evaluating-to-boolean>:

statement

**If Statement**

*This statement is used to control the flow of execution within a program based on some condition*

winner: None

winner is None: True

winner is not None: False

**None Value**

*This is used to represent null values or nothingness*

x+=2 -> x=x+2

x-=2 -> x=x-2

x\*=2 -> x=x\*2

x/=2 -> x=x/2

x//=2 -> x=x//2

x%=2 -> x=x%2

x\*\*=2 -> x=x\*\*2

**Arithmetic Operator**

**Assignment Operator**

Add 1+2 =3

Subtract 5-2 = 3

Multiple 5\*2 =10

Divide 10/2 = 5

Integer Division 7//3 = 2

Modulus 13%3 = 1

Exponent 2\*\*3 = 8

1. X = 2 + 2j
2. Print(‘Real:‘ , X.real , ‘imagn :’ , X.imag)

**Access complex number component**

1. String\_value=’203’
2. Integer\_value=23
4. String\_to\_int=float(String\_value)
5. Float\_to\_int=float(Integer\_value)

**Converting to float**

1. String\_value= ‘200’
2. Float\_value=4.3
4. String\_to\_int = int(String\_value)
5. Float\_to\_int =int(Float\_value)

**Converting to integer**

**Number in Python**

*There are three types number used in python; integer, float and complex number*

**Beginner’s Guide to Python 3 Programming**

**def** my\_function():

a\_variable = 100

print(a\_variable)

**Local Variable**

**Pascal triangle using Recursion**

Pascals triangle is a triangle of the binomial coefficients. The values held in the triangle are generated as follows: In row 0 (the topmost row), there is a unique nonzero entry 1. Each entry of each subsequent row is constructed by adding the number above and to the left with the number above and to the right, treating blank entries as 0

1. lower = int(input("Enter lower number:"))
2. upper = int(input("Enter upper number:"))
4. print("Prime numbers between", lower, "and", upper, "are:")
6. for num in range(lower, upper + 1):
7. if num > 1:
8. for i in range(2, num):
9. if (num % i) == 0:
10. break
11. else:
12. print(num)

**Print prime numbers in range**

1. num=int(input("Enter a number:"))
2. if num>1:
3. for i in range(2,num):
4. if(num%i)==0:
5. print(num,"is not a prime number")
6. break
7. else:
8. print(num,"is a prime number")
10. else:
11. print(num, "is not a prime number")

**Prime number check**

This program able to check if a number is a prime number or not

**Global Keyword**

**def** make\_funtion(s):

**if** s=='add':

**def** adder(x,y):

**return** x+y

**return** adder

**elif** s=='subs':

**def** substarct(x,y):

**return** x-y

**return** substarct

**else**:

**raise** ValueError('Unkown request')

f1=make\_funtion('add')

f2=make\_funtion('subs')

print(f1(3,2))

print(f2(3,2))

**Functions returning functions**

1. def apply(x,function):
2. result=function(x)
3. return result
4. def mult(y):
5. return y\*10.0
6. print(apply(5,mult))

**Simple Example**

**Higher Order Function**

*A function that takes another function as a parameter is known as a higher order function.*

* Functional Programming aims to **avoid side effects**
* Functional Programming **avoids concepts such as state**
* Functional Programming **promotes immutable data**
* Functional Programming **promotes declarative programming**

**Functional Programming Points**

* **Focussed on the computational side** of computer programming
* Generate results based **purely on the data** provided to them
* Functions **only rely on their inputs** to generate a new output
* **Do not generate on any side** **effects** and **do not depend on the current state** of the program

**Functional Programming**

*Is a programming paradigm, a style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids state and mutable data*

def outer():

title = 'original title'

def inner():

nonlocal title title = 'another title' print('inner:', title)

inner()

print('outer:', title)

**nonlocal Keyword**

max = 100

**def** print\_max():

**global** max

max = max + 1

print(max)

**Local and Global Variable**

*“In practice developers usually try to limit the number of global variables in their programs as global variables can be accessed anywhere and can be modified anywhere and this can result in unexpected behaviours (and has been the cause of many, many bugs in all sorts of programs over the years)”*

**Returning multiple value from a Functions**

def swap(a, b):

return b, a

a = 2

b = 3

x, y = swap(a, b)

print(x, ',', y)

**Returning value from a Functions**

def square(n):

return n \* n

**Defining Functions**

def function\_name(parameter list):

"""docstring"""

Statement

statement(s)

**Function in Python**

*Python functions are groups of related statements that can be called together, that typically perform a specific task, and which may or may not take a set of parameters or return a value.*

**Functional Decomposition Terminology**

The key terms used within Functional Decomposition are:

* **Function**. This is a task that is performed by a device, system or process.
* **Decomposition**. This is the process by which higher level functions are broken down into lower level functions where each function represents part of the functionality of the higher level function.
* **Higher Level Function**. This is a function that has one or more sub functions.
* **Sub Function**. This is a function that provides some element of the behaviour of a higher level function.
* **Basic Function**. A basic function is a function that has no smaller sub functions

**Intro Structured Analysis – Functional Decomposition**

*Functional Decomposition is one way in which a system can be broken down into its constituent parts*

**Beginner’s Guide to Python 3 Programming**

**Recursion**

*Recursion is a programming solution which is a function call itself one or more times in order to solve a particular problem*

**Recursive Behaviour**

* The key here is that an overall problem can be solved by breaking it down into smaller examples of the same problem.
* Functions that solve problems by calling themselves are referred to as recursive functions.
* For a recursive function to the useful it must therefore have a termination condition.

**Factorial using Recursion**

**Cek prime number using Recursion**

1. def fact(n):
2. if n==1:
3. return n
4. else:
5. return n\*fact(n-1)

1. def is\_prime(number,a=2):
2. prime=True
3. if a>=number:
4. print("",end='')
5. elif(number%a)==0:
6. prime=False
7. else:
8. return is\_prime(number,a+1)
9. return prime

def pascal\_triangle(n,i=0):

if i>=n:

print("",end="")

else :

#print (i)

for j in range (n-i-1):

print(end=" ")

for j in range (i+1):

print(fact(i)//(fact(j)\*fact(i-j)), end=" ")

print()

pascal\_triangle(n,i+1)

**Class Inheritance**

*Inheritance allows features defined in one class to be inherited and reused in the definition of another class.*

**class** Person:

**def** \_\_init\_\_(self,name,age):

self.name=name

self.age=age

**def** \_\_str\_\_(self):

return self.name+' is '+str(self.age)

**def** birthday(self):

print('Happy birthday you were ',self.age)

self.age+=1

print('You are now', self.age)

**def** calculate\_pay(self,hours\_worked):

rate\_of\_pay=7.50

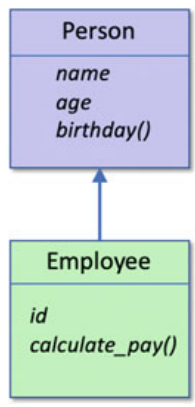
if self.age>=21:

rate\_of\_pay+=2.50

**Beginner’s Guide to Python 3 Programming**

**Example**

That is one class (in this case the Employee class) can inherit features from another class (in this case Person).



**Curried Function**

*A curried function in Python is a function where one or more of its parameters have been applied or bound to a value, resulting in the creation of a new function with one fewer parameters than the original.*

**Class variable**

Referred to as class variable or attributes (as opposed to instance variables or attributes).

1. def multiply(a,b):
2. return a\*b
4. def multby(func,num):
5. return lambda y:func(num,y)
7. double=multby(multiply,2)
8. print(double(5))

1. class Person:
2. instance\_count=0 #class variable
3. def \_\_init\_\_(self,name,age):
4. Person.instance\_count+=1
5. self.name=name
6. self.age=age
8. p1=Person('Jason',36)
9. p2=Person('carol',21)
10. p3=Person('James',19)
11. print(Person.instance\_count)

class A:

def \_\_str\_\_(self):

return 'A'

def print\_info(self):

print('A')

class B:

def \_\_str\_\_(self):

return 'B'

class C:

def \_\_str\_\_(self):

return 'C'

def get\_data(self):

return 'CData'

class D:

def \_\_str\_\_(self):

return 'D'

def print\_info(self):

print('D')

class E:

def \_\_str\_\_(self):

return 'E'

def print\_info(self):

print ('E')

class F(C,D,E):

def \_\_str\_\_(self):

return super().\_\_str\_\_()+'F'

def get\_data(self):

return super().get\_data()+'FData'

def print\_info(self):

print('F'+self.get\_data())

**Multiple Inheritances**

Referred to as class variable or attributes (as opposed to instance variables or attributes).

**class** Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def birthday(self):

print('Happy birthday you were', self.age)

self.age += 1

print('You are now', self.age)

**class Employee(*Person*):**

def \_\_init\_\_(self, name, age, id):

super().\_\_init\_\_(name, age)

self.id = id

def calculate\_pay(self, hours\_worked):

rate\_of\_pay = 7.50

if self.age >= 21:

rate\_of\_pay += 2.50

return hours\_worked \* rate\_of\_pay

**class** Person:

**def \_\_init\_\_**(self,name,age):

self.name=name

self.age=age

***@staticmethod***

**def BMI**(heigh,weight):

heigh=heigh/100

return weight/(heigh\*heigh)

p1=Person('Mario',23)

print(p1.name,' is ',p1.age,', BMI :',Person.BMI(163,52))

p2=Person('Tiara',10)

print(p2.name,' is ', p2.age, ', BMI :',Person.BMI(110,29

**Method**

A method is the name given to behaviour that is linked directly to the Person class; it is not a free-standing function rather it is part of the definition of the class

**Static method**

decorated with the @staticmethod, that is same as free standing function, but is defined within class. These method didn’t other attributes or instances.

c**lass** Person:

instance\_count=0

**@classmethod**

**def** increamnet\_instance\_count(cls):

cls.instance\_count+=1

**def \_\_init\_\_**(self,name,age):

Person.increamnet\_instance\_count()

self.name=name

self.age=age

p1=Person("mario",23)

print(Person.instance\_count)

**Class method**

Decorated with @classmethod and take a first parameter which represents the class rather than an individual instance. Class method linked with class, rather than an individual object. Class method didn’t need an instance, but still need other attributes in the class

**Attributes**

Attributes are instance variable that the class has

**\_\_init\_\_**

\_\_init\_\_ method are local variables and will disappear when the method terminates. This is an initializer (also known as a constructor) for the class. It indicates what data must be supplied when an instance of the Person class is created and how that data is stored internally

**Define a class**

**class** nameOfClass(SuperClass):

\_\_init\_\_

attributes

methods

**Class in Python**

*In Python everything is an object and as such is an example of a type or class of things. For example, integers are an example of the int class, real numbers are.*

*examples of the float class etc*

**Object Orientation**

*OOP provides an approach to structuring programs/applications so that the data held, and the operations performed on that data, are bundled together into classes and accessed via objects*

>>6

>>55

def increment(num):

return num + 1

def reset\_function():

global increment

addition = 50

increment = lambda num: num + addition

print(increment(5))

reset\_function()

print(increment(5))

**Closures**

Closure allows a function to reference a variable available in the scope where the function was originally defined, but not available by default in the scope where it is executed.