Calculator with PyQT and Python

Summer Training Report submitted in partial fulfilment of the requirement for the degree of

 B.Tech

In

Computer Science &Engineering



Training Coordinator By

Ms Mugdha Sharma Prince Yadav

Bhagwan Parshuram Institute of Technology

PSP-4, Sector-17, Rohini, Delhi - 89

November 2020

**DECLARATION**

This is to certify that Report entitled “Calculator with PyQT and Python”which is submitted by me in partial fulfilment of the requirement for the award of degree B.Tech in Computer Engineering to BPIT, GGSIP University, Dwarka, Delhi comprises only my original work and due acknowledgement has been made in the text to all other material used.

Date: Name of Student:

PRINCE YADAV

**ACKNOWLEDGEMENT**

I would like to thank my advisor Mr. Abhijeet Singh for his guidance and patience. Without his insights, feedback, and encouragement, this project would not have been a success.

I would also like to express thanks to Dr. Deepali Verma and Ms Mugdha Sharma. Here I learned how to do programming in python. I am thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staffs to Computer Science Department which helped me in successfully completing my project work.

I would not forget to remember Ms Mugdha Aggarwal, for her encouragement and more over for her timely support and guidance till the completion of my project

**Company Certificate**

# 

**Training Coordinator Certificate**

This is to certify that Report entitled “Programming with Python” which is submitted by Prince Yadav in partial fulfilment of the requirement for the award of degree B.Tech in Computer Engineering  to BPIT, GGSIP University, Dwarka, Delhi is a record of the candidate own work and the matter embodied in this report is adhered to the given format.

Date: Coordinator

  Ms Mugdha Sharma

**TABLE OF CONTENT**

Title page……………………………………………………………………………... I

Declaration……………………………………………………………………………II

Acknowledgement …………………………………………………………………...III

Company certificate …………………………………………………………………IV

Training coordinate certificate ……………………………………………………….V

Abstract ……………………………………………………………………………VIII

Chapter 1(Installing Python)…………………………………………………………..9

Chapter 2(History of Python)………………………………………………………...11

Chapter 3(Introduction to Python)…………………………………………………...12

Chapter 4(Getting Started)…………………………………………………………...15

Chapter 5(Strings)……………………………………………………………………19

Chapter 6 (Set Data Type)……………………………………………………………25

Chapter 7(Python Lists)……………………………………………………………...30

Chapter 8(Tuples)…………………………………………………………………….32

Chapter 9(Dictionary)………………………………………………………………..34

Chapter 10(Principles of OOP)………………………………………………………36

Chapter 11(Developing GUI with PyQT)……………………………………………41

Chapter 12(Application of Python)…………………………………………………..46

Chapter 13(Making the calculator)………………………………………………….48

Bibliography…………………………………………………………………………54

**TABLE OF FIGURES**

|  |  |  |  |
| --- | --- | --- | --- |
| Figure Number | Page Number | Figure Number | Page Number |
| 1.1,1.2 | 9 | 10.3 | 39 |
| 1.3,1.4 | 10 | 11.1 | 42 |
| 2.1 | 11 | 11.2,11.3 | 43 |
| 4.1 | 16 | 11.4,11.5 | 44 |
| 4.2 | 17 | 11.6 | 45 |
| 7.1 | 30 | 13.1,13.2 | 48 |
| 7.2 | 31 | 13.3,13.4 | 49 |
| 9.1,9.2 | 34 | 13.5,13.6 | 50 |
| 9.3,9.4 | 35 | 13.7,13.8 | 51 |
| 10.1 | 37 | 13.9 | 52 |
| 10.2 | 38 | 13.10,13.11 | 53 |

**Name of Table Page Number**

Format Symbol………..……………………………………………………………...21

Common Widgets…………………………………………………………………….41

Major class in PyQT……………………….…………………………………………42

**ABSTRACT**

I have completed my training at Internshala assigned by my college Bhagwan Parshuram Institute of Technology. I choose Python because it has a simple syntax unlike C/C++, Java, and reads almost like plain English, making it is easy to write complex programs. Python falls under the category of High-Level, interpreted languages. Python is an OOP (Object Oriented Programming) based language that gives us benefits such as reducing the number of lines of code, making the code work more efficiently and with minimum errors, that makes programming better and faster. Python comes with many in-built libraries and functions which makes it more understandable. Python comes with a number of choices such as game development, web development, data sciences, operating systems, image processing, and artificial intelligence.First of all, I started my training journey by learning basic python. I prepared my basic python by taking video sessions, studying content given in the form of ppt, and thereby doing assignments related to the topics. After completing it, I focused on basic algorithms and in-built functions.  Then I went through the different data structures available in python for example:: List, Dictionary, Tuple.

Afterwards I have shown how I make the calculator using PyQT and python.

**CHAPTER-1**

**INSTALLING PYTHON**

STEP 1: Go to [www.python.org](http://www.python.org)

Then click on Downloads. Then Windows.

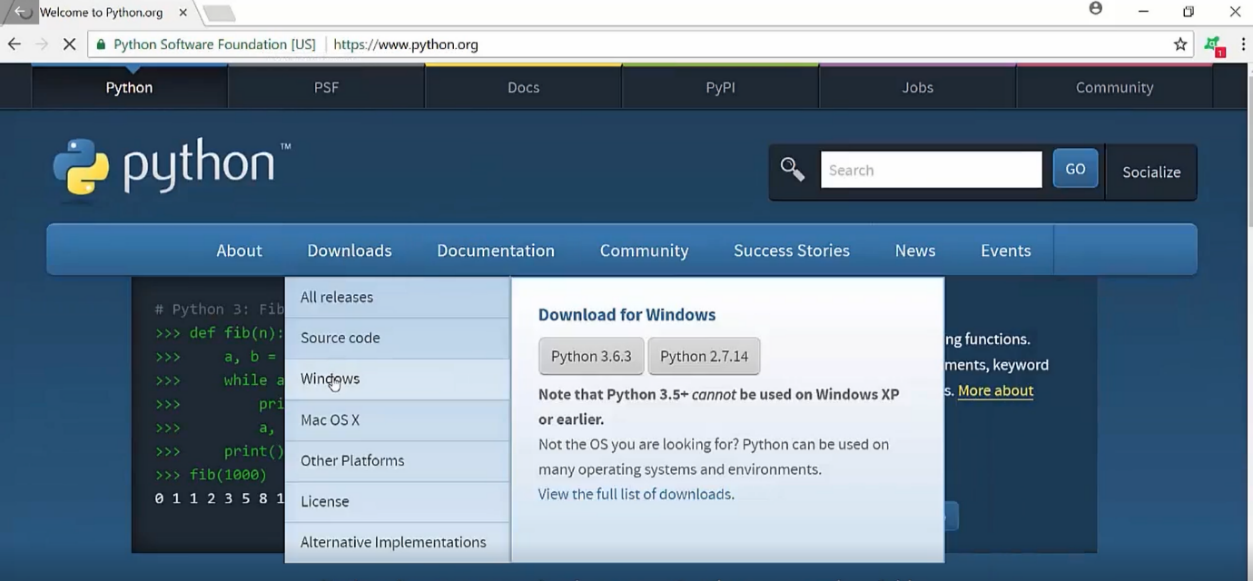


Fig1.1

STEP 2: We used 3.6.2

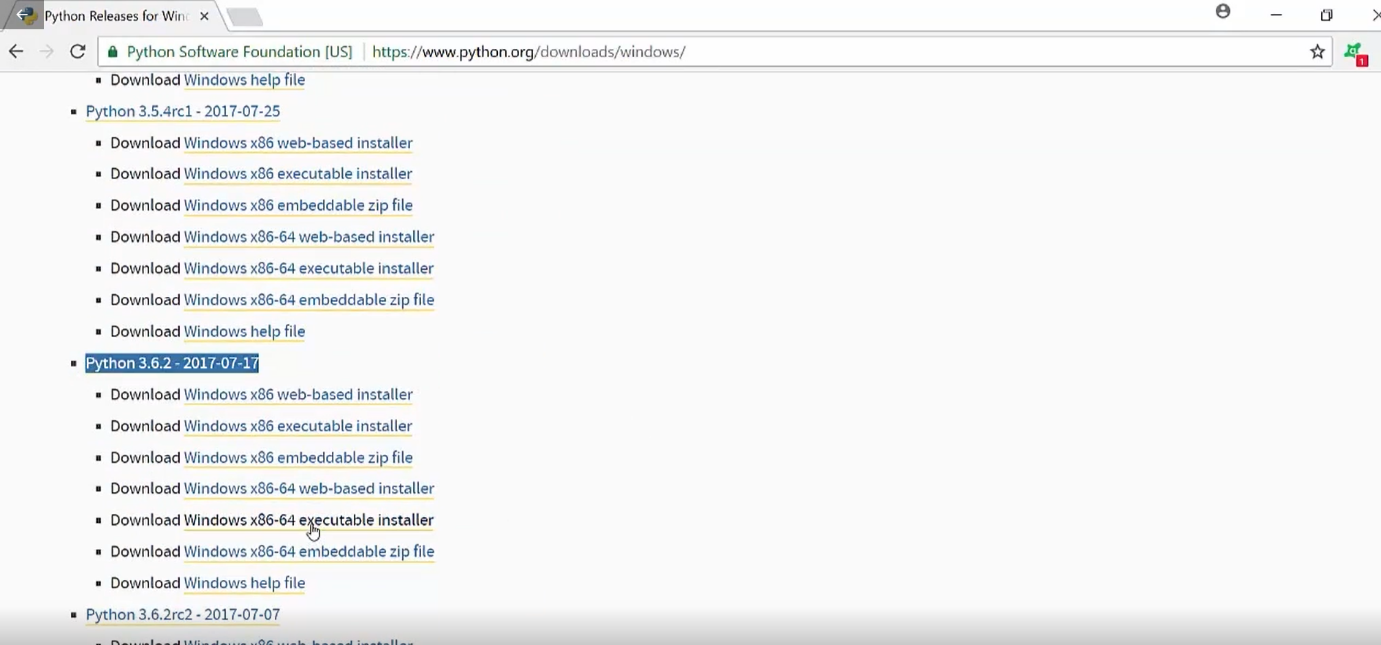


Fig1.2

STEP 3: Open downloaded file.

STEP 4: Do the steps as told by wizard window.



Fig1.3

You can confirm the download by typing python on your command prompt as below.

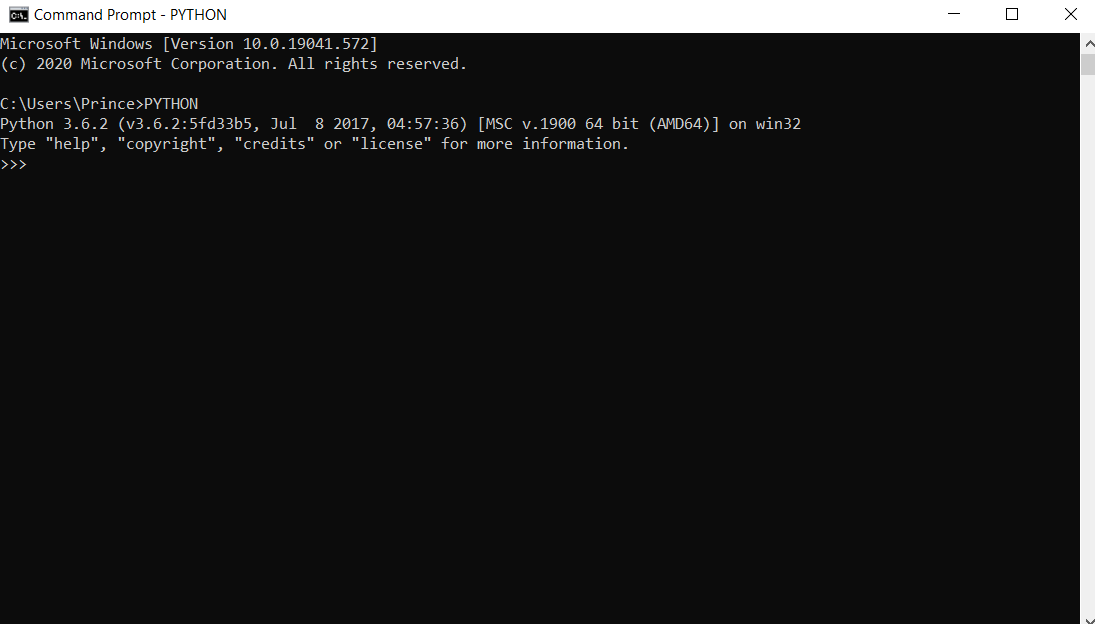


Fig1.4

**CHAPTER-2**

**HISTORY OF PYTHON**

Python is a widely used general-purpose, high-level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code. The programming language which Python is said to have succeeded is ABC Programming Language, which had the interfacing with the Amoeba Operating System and had the feature of exception handling. The language was finally released in 1991. When it was released, it used a lot fewer codes to express the concepts, when we compare it with Java, C++ & C. Its design philosophy was quite good too. Its main objective is to provide code readability and advanced developer productivity. When it was released it had more than enough capability to provide classes with inheritance, several core data types exception handling and functions. The two of the most used versions has to Python 2.x & 3.x. There is a lot of competition between the two and both of them seem to have quite a number of different fan base. For various purposes such as developing, scripting, generation and software testing, this language is utilized. Due to its elegance and simplicity, top technology organizations like Drop box, Google, Quora, Mozilla, Python has been an inspiration for many other coding languages such as Ruby, Cobra, Boo, Coffee Script ECMAScript, Groovy, Swift Go, OCaml, Julia etc. Hewlett-Packard, Qualcomm, IBM, and Cisco have implemented Python.

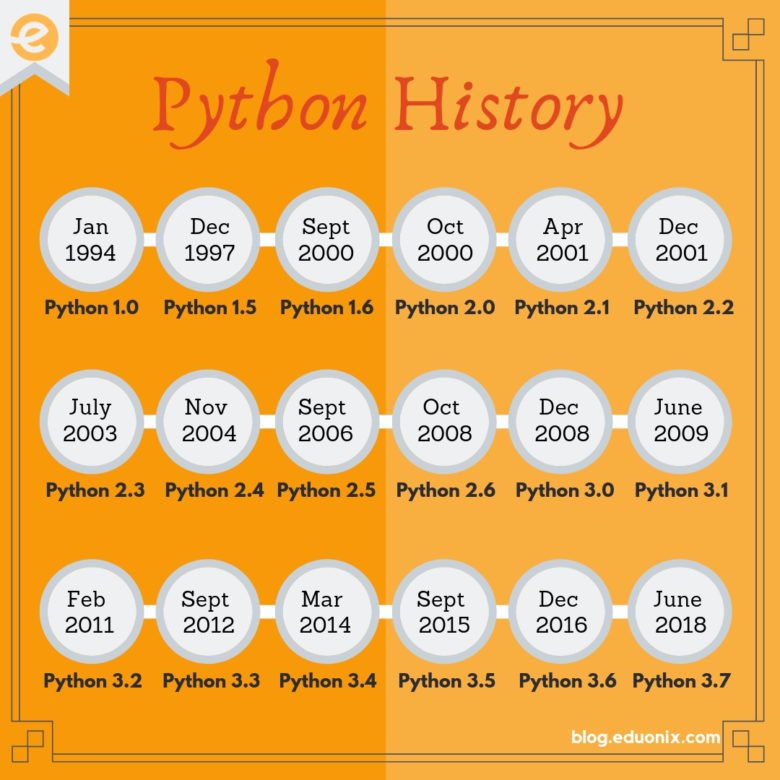


Fig2.1

**CHAPTER -3**

**INTRODUCTION TO PYTHON**

**3.1 SALIENT FEATURES OF PYTHON**

Let us take a closer look at some of the salient features of Python.

*3.1.1Enhanced readability* In Python, uniform indents are used to delimit blocks of statements instead of curly brackets like in C, C++ and Java.

This enhances readability. In C, C++, Java

|  |
| --- |
| if (X%2==0) {  if (x%3==0)  System.out.println("divisible by 2 and 3");  else  System.out.println("divisible by 2 not divisible by 3");  }  else  { if (x%3==0)  System.out.println("divisible by 3 not divisible by 2");  Else  System.out.println("not divisible by 3 nor by 2"); } |

In Python

|  |
| --- |
| if num%2 == 0:  if num%3 == 0:  print ("Divisible by 3 and 2")  else: print ("Divisible by 2 not divisible by 3")  else:  if num%3 == 0:  print ("Divisible by 3 not divisible by 2")  else:  print ("Not divisible by 3 nor by 2") |

* + 1. *Dynamic typing:*

In Python, a variable to be used in a program need not have prior declaration of its name and type.

* + 1. *Interpreted language:*

Python is an interpreter-based language. An interpreter executes one instruction at a time. So if there is an error on line 7 of the code, it will execute instructions till line 6 and then stop. This is unlike a compiler which will not execute any of the instructions even if there is a single error. Thus, an interpreter is useful if you are new to programming because it allows you to see partial output of your code and identify the error location more easily.

* + 1. *Extensible language:*

Python extension modules implement new built-in object types and can call C libraries and system calls which Python doesn’t have direct access to.

* + 1. *Standard DB2 API:*

A standard data connectivity API facilitates using data sources such as Oracle, MySQL and SQLite as a backend to a Python program for storage, retrieval and processing of data.

* + 1. *GUI programming:*

Standard distribution of Python contains Tkinter GUI kit, which is an implementation of the popular GUI library called Tcl/Tk. An attractive GUI can be constructed using Tkinter. Many other GUI libraries like Qt, GTK, WxWidgets etc. are also ported to Python.

* + 1. *Embeddable:*

Python can be integrated with other popular programming technologies such as C, C++, Java, ActiveX and CORBA.

**1.2 COMPARING PYTHON 2.x AND PYTHON 3.x**

Although development of Python started in the late 80’s, Python 1.0 was published in  January 1994. Some important features like cycle-detecting garbage collector and  Unicode support were added in Python 2.0 which was released in October 2000. Python 3.0 was released in December 2008. It was designed to rectify certain flaws in  earlier version. The guiding principle of Python 3 was: "reduce feature duplication by  removing old ways of doing things". Python 3.0 doesn’t provide backward  compatibility. That means a Python program written using version 2.x syntax doesn’t  execute under python 3.x interpreter. Since all the Python libraries are not fully ported  to Python 3, Python Software Foundation continues to support Python 2. The  foundation has recently announced it will discontinue Python 2 support by year 2020.

*KEY DIFFERENCE*

• Python 3 syntax is simpler and easily understandable whereas Python 2 syntax  is comparatively difficult to understand.

• Python 3 default storing of strings is Unicode whereas Python 2 stores need to  define Unicode string value with "u."

• Python 3 value of variables never changes whereas in Python 2 value ofthe  global variable will be changed while using it inside forloop.

• Python 3 exceptions should be enclosed in parenthesis while Python 2  exceptions should be enclosed in notations.

• Python 3 rules of ordering comparisons are simplified whereas Python 2 rules  of ordering comparison are complex.

• Python 3 offers Range() function to perform iterations whereas, In Python 2,  the xrange() is used for iterations.

**CHAPTER-4**

**GETTING STARTED**

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde &

Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired

by SETL), capable of exception handling and interfacing with the Amoeba operating

system. Its implementation began in December 1989. Python is a high-level language

intended to be relatively straightforward for humans to read and write and for computers

to read and process. Python is an interpreter and when we are running Python

interactively, we can type a line of Python (a sentence) and Python processes it

immediately and is ready for us to type another line of Python.

Reserved words

**4.1 COMMON KEYWORDS USED IN PYTHON**

'False' 'elif'

'lambda' 'None'

'else' 'nonlocal'

'True' 'except'

'not' 'and'

'finally' 'or'

'as' 'for'

'pass' 'assert'

'from' 'raise'

'break' 'global'

'return' 'class'

'if' 'try'

'continue' 'import'

'while' 'def'

'in' 'with'

'del' 'is'

'yield’

**4.2 Data Types in Python**

The built-in data types in Python are displayed below

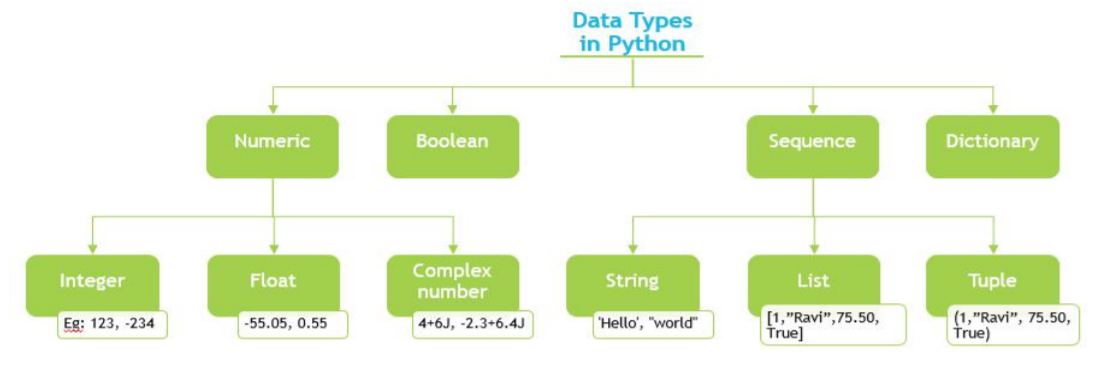


Fig4.1

*4.2.1 Numeric:*

Any representation of data which has numeric value. Python identifies three types of numbers – integer, float and complex number. Integer Positive and negative whole numbers.

Examples: 1234, -234, 0x46 (hexadecimal number), 0O123 (octal number))

( Note: In C and related programming languages such as Python, a hexadecimal number is prefixed with 0x and an octal number is prefixed with 0O.)

Float:

*Real numbers* with a floating point representation in which the fractional component is denoted by a decimal or scientific notation

Examples: -55.550, 0.005, 1.32E10 (scientific notation))

*Complex number:*  A number with a real and imaginary component is represented as a + bj inPython where a and b are floats and j = √-1 Examples: 4+6j, -2.3+6.4j

(Note: The common mathematical representation of a complex number uses a +bi with i being the imaginary part. But in electronics j is used because i already represent current and the next letter after i is j.)

*4.2.2 Boolean:*

Any representation of data which has two values denoted by True and False.

Sequence:

An ordered collection of similar or different data types. The built-in Sequence data types in Python are – String, List and Tuple.

1. String- A collection of one or more characters put in single, double or triple quotes. Examples: ‘Hello’, "Hello", "'Hello'", """Hello"""

2. List- An ordered collection of one or more data items, not necessarily of same type, put in square brackets. Examples: [1,"Ravi",75.50, True]

3. Tuple- An ordered collection of one or more data items, not necessarily of same type put in parentheses. The contents of a tuple cannot be modified – it is immutable - after the tuple is created. Examples: (1,"Ravi", 75.50, True) Note: Refer to the Helper Text to learn more about mutability.

*4.2.3 Dictionary:*

An unordered collection of data in key:value pair form. Collection of such pairs is enclosed in curly brackets. Example: {1:"Superman", 2:"Wonder Woman", 3:"Thor", 4: "Hulk", 5:"Black Widow"}

**4.3 Mutable and Immutable Objects**

When a program is run, data objects in the program are stored in the computer’s memory for processing. While some of these objects can be modified at that memory location, other data objects can’t be modified once they are stored in the memory.

The property of whether or not data objects can be modified in the same memory location where they are stored is called mutability.

We can check the mutability of an object by checking its memory location before and after it is modified. If the memory location remains the same when the data object is modified, it means it is mutable.

To check the memory location of where a data object is stored, we use the function, id()



Fig4.2

Notice that the memory location has not changed as the ID remains (1906292064) remains the same before and after the variable is modified. This indicates that the list is mutable, i.e., it can be modified at the same memory location where it is stored.

*Immutable:* numeric, string, and tuple

*Mutable:* list, dictionary

**4.4 Variables**

Variables are own created words that have meaning to user called *variables*. You will  have great latitude in choosing your names for your variables, but you cannot use and of  Python’s reserved words as a name for a variable. These do not required to the declare  the type of these variables before use we can declare these any where through out the  program.

**4.5 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.

**4.6 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.

Precedence order for operators are:-

Paranthesis>Exponent>Multiplication=Division>Addition=Subtraction

The *modulus operator* works on integers and yields the remainder when the firstoperand  is divided by the second.Use a % sign to as modulus operator.

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.

**4.7 Types of error**

*4.7.1 Syntax errors*

A syntax error means that you have violated the “grammar” rules of Python. Python does  its best to point right at the line and character where it noticed it was confused.

*4.7.2 Logic errors*

A logic error is when your program has good syntax but there is a mistake in the order of  the statements or perhaps a mistake in how the statements relate to one another.

*4.7.3 Semantic errors*

A semantic error is when your description of the steps to take is syntactically perfect and

in the right order, but there is simply a mistake in the program. The program is perfectly

correct but it does not do what you intended for it to do.

**CHAPTER-5**

**STRINGS**

A string is a sequence of characters. You can access the characters one at a time with the bracket operator.

>>> fruit = 'banana'

>>> letter = fruit[1]

>>> a

The second statement extracts the character at index position 1 and expression in brackets is called an

index. Also the indices starts from 0 and is integer ,also -1 gives last letter, -2 gives second last letter and so

on.

**5.1 String Operators**

You can use certain operators for string processing.

*5.1.1 + Concatenation*

[+] Appends the second string to the first.

Example

>>> a='"Hello, ' #String a

>>> b='Nick",' #String b

>>> print(a+b+ ' said Harry to Nearly Headless Nick.') #The + here appends the strings a and b to the string in quotes.

"Hello, Nick", said Harry to Nearly Headless Nick.

>>>

*5.1.2 + Repetition*

[\*] Concatenates multiple copies of the same string.

Example

>>> a='I must not tell lies…' # String a

>>> print('Again and again Harry wrote the words on the parchment in his own blood – '+a\*5) # Here, the \* joins or concatenates the string a repeatedly for 5 times.Again and again Harry wrote the words on the parchment in his own blood – I must not tell lies…I must not tell lies…I must not tell lies…I must not tell lies…I must not tell lies…

>>>

*5.1.3 [] Slice []*

Gives the character at given index.

Example

T h e B u r r o w

0 1 2 3 4 5 6 7 8 9

>>> a='The Burrow' #String a

>>> a[8] #Here the index number in square brackets [ ] slices out the character at that index, which is o in this example.'o'

>>>

*5.1.4 [:] Range Slice*

[ : ] Fetches characters in the range specified by two index operands separated by a colon.

If the first operand is omitted, the range starts at zero index.

If the second operand is omitted, the range goes up to the end of the string.

Note: The slice starts at the first index. The slice ends one index before the second index, that is at the value of the index - 1.

Example

T h e B u r r o w

0 1 2 3 4 5 6 7 8 9

>>> a='The Burrow' #String a

>>> a[2:7] #Starting index = 2 = e, Ending index = 7-1 = r'e Bur'

>>> a[:6] #Starting index = 0 = T, Ending index = 6-1 = u'The Bu

'>>> a[5:] #Starting index = 5 = u, Ending index = end of string = w'urrow'

>>>

*5.1.5 in Membership*

[in] Returns true if a character exists in the given string.

Example

>>> a='Harry watched Dumbledore striding up and down in front of him, and thought. He thought of his mother, his father and Sirius. He thought of Cedric Diggory.' #String a

>>> 'v' in a #Checks if the character 'v' is present in the string, aFalse

>>> 'dig' in a #Checks if the characters 'dig' are present in the string, aFalse

>>> 'Dig' in a #Note that this is case-sensitive. As 'Dig' is present in 'Diggory', this returns True.True

>>>

*5.1.6 not in Membership*

[not in] Returns true if a character does not exist in the given string.

Example

>>> a=''' “For HIM?” shouted Snape. “Expecto Patronum!"

From the tip of his wand burst the silver doe: She landed on the office floor, bounded once across the office, and soared out of the window. Dumbledore watched her fly away, and as her silvery glow faded he turned back to Snape, and his eyes were full of tears.

“After all this time?”

“Always,” said Snape.''' #Multi-line string, a

>>> 'v' not in a #Checks that the character 'v' is not present in the string, a False

>>> 'Red' not in a #Checks that the characters 'Red' iare not present in the string, a True

>>> 'red' notin a #This is case sensitive. Since 'red' is present in 'soared', this returns False. False

**5.2Format specification symbols**

Format Symbol Conversion

%c character Example 1

>>> balltype='basketball'

>>> result='hit'

>>> print('I wondered why the %s was

getting bigger. Then it %s me.' %

(balltype, result))

I wondered why the basketball was

getting bigger. Then it hit me.

%s string conversion via str() Example 2

prior to formatting >>> print("%20s" % ('Internshala', ))

Internshala

>>>

Example 3

>>> print("%-20s" % ('Internshala', ))

Internshala

>>>

Example 4

>>>

print("%.5s" % ('Internshala', ))

Inter

>>>

%i signed decimal integer

%d signed decimal integer

>>> match=12553

>>> site='eBay'

>>> print("%s is so useless. I tried to

look up lighters and all they had

was %d matches." % (site, match))

eBay is so useless. I tried to look up

lighters and all they had was 12553

matches.

>>>

%u unsigned decimal integer

%o octal integer

**5.3 The format() Method**

With Python 3.0, the format() method has been introduced for handling complex string formatting more efficiently. This method of the built-in string class provides functionality for complex variable substitutions and value formatting. This new formatting technique is regarded as more elegant. The general syntax of format() method is:

string.format(var1, var2,...)

The string itself contains placeholders {} in which values of variables are successively inserted.

|  |
| --- |
| >>> name="Malhar"  >>> age=23  >>> percentage=55.5  >>>"my name is {} and my age is {} years".format(name, age)  'my name is Malhar and my age is 23 years'  >>> |

You can also specify formatting symbols. Only change is using colon (:) instead of %. For example, instead of %s use {:s} and instead of %d use (:d}

|  |
| --- |
| >>> "my name is {:s} and my age is {:d} years".format(name, age)  'my name is Malhar and my age is 23 years'  >>> |

Precision formatting of numbers can be accordingly done

.

|  |
| --- |
| >>> "my name is {:s}, age {:d} and I have scored {:6.3f} percent marks".format(name, age, percentage)  'my name is Malhar, age 23 and I have scored 55.500 percent marks'  >>> |

**5.4 Other Methods for String Processing**

*5.4.1 capitalize():* This method converts the first character of a string to uppercase letter.

|  |
| --- |
| >>> var='internshala'  >>> var.capitalize()  'Internshala'  >>> |

*5.4.2 upper():* This method returns a string with lowercase characters replaced by corresponding uppercase characters.

|  |
| --- |
| >>> var='internshala'  >>> var.upper()  'INTERNSHALA'  >>> |

*5.4.3 lower():* This method results in a string with uppercase characters replaced by corresponding lowercase characters.

|  |
| --- |
| >>> var='INTERNSHALA'  >>> var.lower()  'internshala'  >>> |

*5.4.4 title():* This method results in a string with the first character of each word converted to uppercase.

|  |
| --- |
| >>> var='python training from internshala'  >>> var.title()  'Python Training From Internshala'  >>> |

*5.4.5 find():* This method finds the first occurrence of a substring in another string. If not found, the method returns -1.

|  |
| --- |
| >>> var='python training from internshala'  >>> var.find('in')  10  >>> var.find('on')  4  >>> var.find('run')  -1  >>> |

The substring 'in' first occurs at the 10 th position (count starts from 0), 'on' is found at the 4 th position, but 'run' is not found hence returns -1.

*5.4.6 index():* This method is similar to find() but throws a ValueError if the substring is not found.

|  |
| --- |
| >>> var='python training from internshala'  >>> var.index('in')  10  >>> var.index('run')Traceback (most recent call last):  File "<pyshell#19", line 1, in<module>  var.index('run')  ValueError: substring not found  >>> |

*5.4.7 count():* This method returns the number of occurrences of a substring in given string

|  |
| --- |
| >>> var='python training from internshala'  >>> var.count('in')  3  >>> |

*5.4.8 isalpha():* This method returns true if all the characters in a string are alphabetic (a-z or A-Z), otherwise returns false.

|  |
| --- |
| >>> var='Internshala'  >>> var.isalpha()  True  >>> var='Intern shala'  >>> var.isalpha()  False  >>> |

*5.4.9 isdigit():* This method returns true if all characters in a string are digits( 0-9), if not returns false.

|  |
| --- |
| >>> var='2000'  >>> var.isdigit()  True  >>> var='2,000'  >>> var.isdigit()  False  >>> |

*5.4.10 islower():* This method returns true if all characters in a string are lowercase characters else returns false.

|  |
| --- |
| >>> var='internshala'  >>> var.islower()  True  >>> var='Internshala'  >>> var.islower()  False  >>> var='intern shala'  >>> var.islower()  True  >>> |

*5.4.11 isupper():* This method returns true if all characters in a string are uppercase characters else returns false.

|  |
| --- |
| >>> var='INTERN\_SHALA'  >>> var.isupper()  True  >>> var='INTERNshala'  >>> var.isupper()  False  >>> var='INTERN+SHALA'  >>> var.isupper()  True  >>> var='1234'  >>>  var.isupper()  False |

**CHAPTER-6**

**SET DATA TYPE**

Set is also a collection data type in Python. However, it is not an ordered collection of objects, like list or tuple. Hence, indexing and slicing operations cannot be done on a set object. A set also doesn’t allow duplicate objects to be stored, where as in list and tuple, the same object can appear more than once. Even if an object is put more than once in a set, only one copy is held. Set is a Python implementation of a set as defined in Mathematics. The set object has suitable methods to perform mathematical set operations like union, intersection, difference etc. A set object contains one or more items, not necessarily of the same type which are separated by comma and enclosed in curly brackets {}

|  |
| --- |
| >>> S1={1, "Ravi", 75.50}  >>> S1 {1, 75.5, 'Ravi'}  >>> type(S1)  <class’set’  >>> S2={10,23,40,23,50,10}  >>>  S2 {40, 10, 50, 23}  >>> |

**6.1 set() function**

Python has an in-built function set() using which set object can be constructed out of any sequence like string, list or tuple object.

|  |
| --- |
| >>> S1=set("Internshala")  >>> S1  {'t', 'n', 's', 'h', 'e', 'a', 'l', 'I', 'r'}  >>> S2=set([45,67,87,36,55])  >>> S2  {55, 67, 36, 45, 87}  >>> S3=set((10,25,15))  >>> S3  {25, 10, 15}  >>> |

Order of elements in the set is not necessarily the same that is given at the time of assignment. Python optimizes the structure for performing operations over set as defined in mathematics. Only immutable (and hashable) objects can be a part of set object. Numbers (integer, float as well as complex), strings, and tuple objects are accepted but list and dictionary objects are not.

|  |
| --- |
| >>> S1={(10,10), 10,20}  >>> S1  {10, 20, (10, 10)}  >>> S2={[10,10], 10,20}  Traceback (most recent call last):  File "pyshell#2", line 1, in<module>  S2={[10,10], 10,20}  TypeError: unhashable type: 'list'  >>> |

In first case, (10,10) is a tuple, hence it becomes part of set. In second example though, since [10,10] is a list, error message is displayed saying list is unhashable. (Hashing is a mechanism in computer science which enables quicker searching of objects in computer’s memory. https://en.wikipedia.org/wiki/Hash\_function) Even though mutable objects are not stored in a set, set itself is a mutable object.

**6.2** A set object can be modified by add(), update(), remove() and discard() methods.

*6.2.1 add()*

adds a new element in set object

|  |
| --- |
| >>> S1=set({"Python", "Java", "C++"})  >>> S1 {'Python', 'C++', 'Java'}  >>> S1.add("Perl")  >>> S1  {'Perl', 'Python', 'C++', 'Java'}  >>> |

*6.2.2update()*

adds multiple items from a list or tuple

|  |
| --- |
| >>> S1={"Python", "Java", "C++"}  >>> S1.update(["C++", "Basic"])  >>> S1  {'C++', 'Java', 'Python', 'Basic'}  >>> S1.update(["Ruby", "PHP"])  >>>  S1 {'Ruby', 'PHP', 'Java', 'C++', 'Python', 'Basic'}  >>> |

*6.2.3clear()*

Removes contents of set object results in an empty set

|  |
| --- |
| >>> S1.clear()  >>> S1  set() |

*6.2.4copy()*

Creates a copy of set object

|  |
| --- |
| >>> S1={"Python", "Java", "C++"}  >>> S2=S1.copy()  >>> S2 {'C++', 'Java', 'Python'}  >>> |

*6.2.5 discard()*

Returns set after removing an item from it. No changes are done if the item is not present

|  |
| --- |
| >>> S1={"Python", "Java", "C++"}  >>> S1.discard("Java")  >>> S1  {'C++', 'Python'}  >>> S1.discard("SQL")  >>> S1  {'C++', 'Python'}  >>> |

*6.2.6 remove()*

Returns set after removing an item from it. Results in error if the item is not present

|  |
| --- |
| >>> S1={"Python", "Java", "C++"}  >>> S1.remove("C++")  >>> S1  {'Java', 'Python'}  >>> S1.remove("SQL")  Traceback (most recent call last):  File "pyshell#3", line 1, in <module>  S1.remove("SQL")  KeyError: 'SQL'  >>> |

**6.3 Set Operations**

As mentioned earlier, set data type in Python implements set as defined in mathematics. Various Set operations can be performed using Python’s set obect. The operators |, &, - and ^ perform union, intersection, difference and symmetric difference operations respectively. Each of these operators have a corresponding method associated with built-in set class.

*6.3.1 Union*

Union of two sets is a set of all elements in both.

|  |
| --- |
| >>> s1={1,2,3,4,5}  >>> s2={4,5,6,7,8}  >>> s1|s2  {1, 2, 3, 4, 5, 6, 7, 8}  >>> s1.union(s2)  {1, 2, 3, 4, 5, 6, 7, 8}  >>> |

*6.3.2 Intersection*

Intersection of two sets is a set containing elements common to both

|  |
| --- |
| >>> s1={1,2,3,4,5}  >>> s2={4,5,6,7,8}  >>> s1&s2 {4, 5}  >>> s1.intersection(s2)  {4, 5}  >>> |

*6.3.3 Difference*

Difference of two sets results in a set containing elements only in first but not in second set.

|  |
| --- |
| >>> s1={1,2,3,4,5}  >>> s2={4,5,6,7,8}  >>> s1-s2  {1, 2, 3}  >>> s2-s1  {8, 6, 7}  >>> s1.difference(s2)  {1, 2, 3}  >>> s2.difference(s1)  {8, 6, 7}  >>> |

*6.4.4 Symmetric Difference*

Result of Symmetric difference is a set consisting of elements in both sets excluding common elements

|  |
| --- |
| >>> s1={1,2,3,4,5}  >>> s2={4,5,6,7,8}  >>> s1^s2  {1, 2, 3, 6, 7, 8}  >>> s2^s1  {1, 2, 3, 6, 7, 8}  >>> s1.symmetric\_difference(s2)  {1, 2, 3, 6, 7, 8}  >>> s2.symmetric\_difference(s1)  {1, 2, 3, 6, 7, 8}  >>> |

Set is a specialized data type. One of the major applications of Python is in area of mathematical computing and data analysis in which set operations are important. We may drop this discussion considering it as not for beginner (and also to curtail the size), but learner (especially who intends to go in mathematical and scientific computing) should be encouraged to explore this section. We should emphasize this and provide this as a text for further reading.

**CHAPTER-7**

**PYTHON LISTS**

In short, a list is a collection of arbitrary objects, somewhat akin to an array in many other programming languages but more flexible. Lists are defined in Python by enclosing a comma-separated sequence of objects in square brackets ([]), as shown below:

>>>

>>> a = ['foo', 'bar', 'baz', 'qux']

>>> print(a)

['foo', 'bar', 'baz', 'qux']

>>> a

['foo', 'bar', 'baz', 'qux']

The important characteristics of Python lists are as follows:

* Lists are ordered.
* Lists can contain any arbitrary objects.
* List elements can be accessed by index.
* Lists can be nested to arbitrary depth.
* Lists are mutable.
* Lists are dynamic.
* Individual elements in a list can be accessed using an index in square brackets. This is exactly analogous to accessing individual characters in a string. List indexing is zero-based as it is with strings.
* Consider the following list:

>>>

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

The indices for the elements in a are shown below:

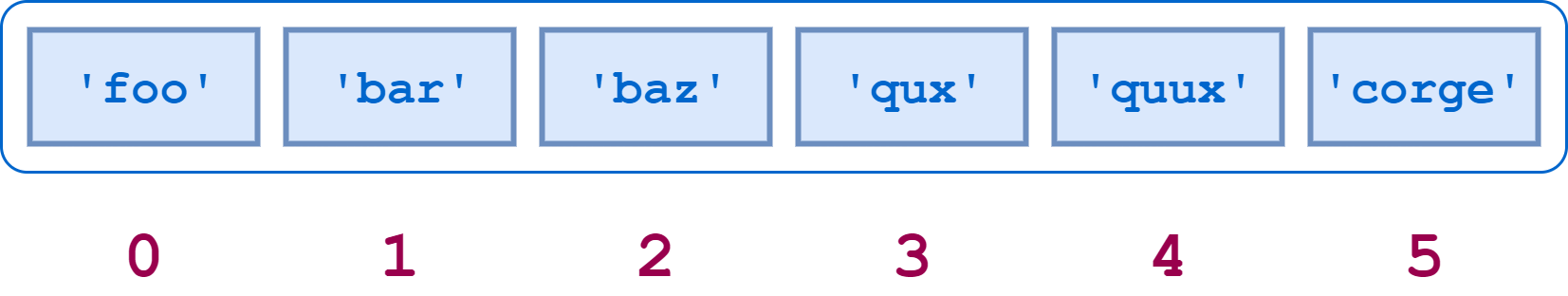
[](https://files.realpython.com/media/t.eb0b38e642c5.png)

Fig7.1

List Indices

Here is Python code to access some elements of a:

>>>

>>> a[0]

'foo'

>>> a[2]

'baz'

>>> a[5]

'corge'

Virtually everything about string indexing works similarly for lists. For example, a negative list index counts from the end of the list:

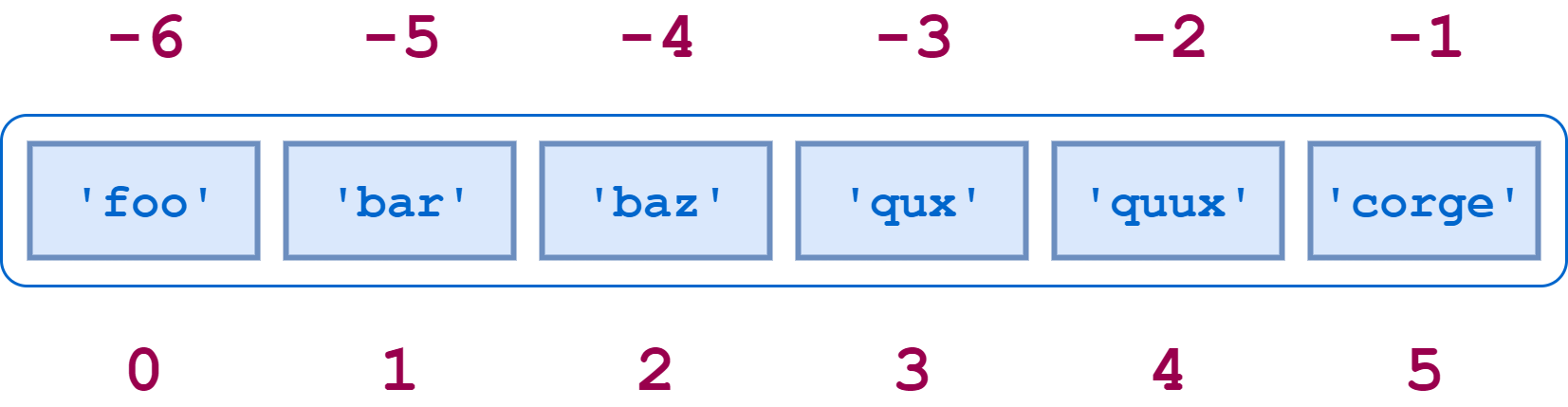
[](https://files.realpython.com/media/t.c11ea56e8ca2.png)

Fig7.2

Negative List Indexing

### **CHAPTER-8**

### **TUPLES**

Tuples are identical to lists in all respects, except for the following properties:

* Tuples are defined by enclosing the elements in parentheses (()) instead of square brackets ([]).
* Tuples are immutable.

Here is a short example showing a tuple definition, indexing, and slicing:

>>>

>>> t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')

>>> t

('foo', 'bar', 'baz', 'qux', 'quux', 'corge')

>>> t[0]

'foo'

>>> t[-1]

'corge'

>>> t[1::2]

('bar', 'qux', 'corge')

Never fear! Our favorite string and list reversal mechanism works for tuples as well:

>>>

>>> t[::-1]

('corge', 'quux', 'qux', 'baz', 'bar', 'foo')

**Note:** Even though tuples are defined using parentheses, you still index and slice tuples using square brackets, just as for strings and lists.

Everything you’ve learned about lists—they are ordered, they can contain arbitrary objects, they can be indexed and sliced, they can be nested—is true of tuples as well. But they can’t be modified:

>>>

>>> t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')

>>> t[2] = 'Bark!'

Traceback (most recent call last):

File "<pyshell#65>", line 1, in <module>

t[2] = 'Bark!'

TypeError: 'tuple' object does not support item assignment

Why use a tuple instead of a list?

* Program execution is faster when manipulating a tuple than it is for the equivalent list. (This is probably not going to be noticeable when the list or tuple is small.)
* Sometimes you don’t want data to be modified. If the values in the collection are meant to remain constant for the life of the program, using a tuple instead of a list guards against accidental modification.
* There is another Python data type that you will encounter shortly called a dictionary, which requires as one of its components a value that is of an immutable type. A tuple can be used for this purpose, whereas a list can’t be.

In a Python REPL session, you can display the values of several objects simultaneously by entering them directly at the >>> prompt, separated by commas:

>>>

>>> a = 'foo'

>>> b = 42

>>> a, 3.14159, b

('foo', 3.14159, 42)

Python displays the response in parentheses because it is implicitly interpreting the input as a tuple.

There is one peculiarity regarding tuple definition that you should be aware of. There is no ambiguity when defining an empty tuple, nor one with two or more elements. Python knows you are defining a tuple:

>>>

>>> t = ()

>>> type(t)

<class 'tuple'>

>>>

>>> t = (1, 2)

>>> type(t)

<class 'tuple'>

>>> t = (1, 2, 3, 4, 5)

>>> type(t)

<class 'tuple'>

**CHAPTER-9**

**DICTIONARY**

Another useful data type built into Python is the dictionary. Dictionaries are

sometimes found in other languages as “associative memories” or “associative

arrays”. Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.

Lists can’t be used as keys, since lists can be modified in place using index

assignments, slice assignments, or methods like append() and extend(). It is best to think of a dictionary as a set of key: value pairs, with the requirement that the keys are unique (within one dictionary).

A pair of braces creates an empty dictionary: {}. Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output. The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key:value pair with del.

Performing list(d) on a dictionary returns a list of all the keys used in the dictionary, in insertion order (if you want it sorted, just use sorted(d) instead). To check whether a single key is in the dictionary, use the in keyword.

Here is a small example using a dictionary:

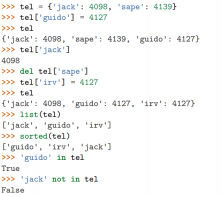


Fig9.1

The dict() constructor builds dictionaries directly from sequences of key-value pairs:



Fig9.2

In addition, dict comprehensions can be used to create dictionaries from arbitrary key and value expressions:



Fig9.3

When the keys are simple strings, it is sometimes easier to specify pairs using

keyword arguments:



Fig9.4

**CHAPTER-10**

**PRINCIPLES OF OBJECT-ORIENTED PROGRAMMING(OOP)**

Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy.

Here is small introduction of Object-Oriented Programming (OOP) to bring you at speed

**10.1 OOP TERMINOLOGY**

*10.1.1 Class* − A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

*10.1.2 Class variable* − A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.

*10.1.3 Data member* − A class variable or instance variable that holds data associated with a class and its objects.

*10.1.4 Function overloading* − The assignment of more than one behaviour to a particular function. The operation performed varies by the types of objects or

arguments involved.

*10.1.5 Instance variable* − A variable that is defined inside a method and belongs only to the current instance of a class.

*10.1.6 Inheritance* − The transfer of the characteristics of a class to other classes that are derived from it.

*10.1.7 Instance* − An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.

*10.1.8 Instantiation* − The creation of an instance of a class.

*10.1.9 Method* − A special kind of function that is defined in a class definition.

*10.1.10 Object* − A unique instance of a data structure that's defined by its class. An

object comprises both data members (class variables and instance variables) and methods.

*10.1.11 Operator overloading* − The assignment of more than one function to a particular operator.

**10.2 CREATING CLASSES**

The class statement creates a new class definition. The name of the class immediately

follows the keyword class followed by a colon as follows –

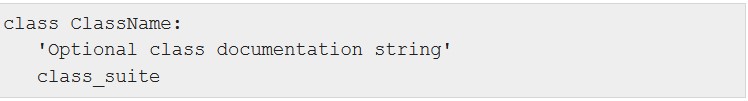


Fig10.1

• The class has a documentation string, which can be accessed via

ClassName.\_\_doc\_\_.

• The class\_suite consists of all the component statements defining class

members, data attributes and functions.

**10.3 CREATING INSTANCE OBJECTS**

To create instances of a class, you call the class using class name and pass in whatever arguments its \_\_init\_\_ method accepts.

**10.4 ACCESSING ATTRIBUTES**

You access the object's attributes using the dot operator with object. Class variable would be accessed using following functions −

• The getattr(obj, name[, default]) − to access the attribute of object.

• The hasattr(obj,name) − to check if an attribute exists or not.

• The setattr(obj,name,value) − to set an attribute. If attribute does not exist,

then it would be created.

• The delattr(obj, name) − to delete an attribute.

**10.5 BUILT-IN CLASS ATTRIBUTES**

Every Python class keeps following built-in attributes and they can be accessed using

dot operator like any other attribute − • \_\_dict\_\_ − Dictionary containing the class's

namespace.

• \_\_doc\_\_ − Class documentation string or none, if undefined.

• \_\_name\_\_ − Class name.

• \_\_module\_\_ − Module name in which the class is defined. This attribute is

"\_\_main\_\_" in interactive mode.

• \_\_bases\_\_ − A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

**10.6 DESTROYING OBJECTS** (GARBAGE COLLECTION)

Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space. The process by which Python periodically reclaims blocks of memory that no longer are in use is termed Garbage Collection.

Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero. An object's reference count changes as the number of aliases that point to it changes. An object's reference count increases when it is assigned a new name or placed in a container (list, tuple, or dictionary). The object's reference count decreases when it's deleted with del, its reference is reassigned, or its reference goes out of scope. When an object's reference count reaches zero, Python collects it automatically.

You normally will not notice when the garbage collector destroys an orphaned instance and reclaims its space. But a class can implement the special method \_\_del\_\_(), called a destructor, that is invoked when the instance is about to be destroyed. This method might be used to clean up any non memory resources used by an instance.

This \_\_del\_\_() destructor prints the class name of an instance that is about to be destroyed .

**10.7 CLASS INHERITANCE**

Instead of starting from scratch, you can create a class by deriving it from a pre existing class by listing the parent clas cc s in parentheses after the new class name.

The child class inherits the attributes of its parent class, and you can use those

attributes as if they were defined in the child class. A child class can also override data members and methods from the parent.

*Syntax*

Derived classes are declared much like their parent class; however, a list of base classes to inherit from is given after the class name –

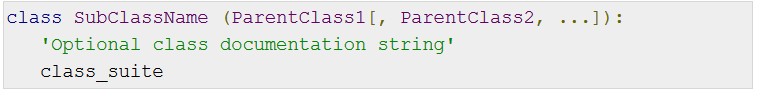


Fig10.2

You can use issubclass() or isinstance() functions to check a relationships of two classes and instances.

• The issubclass(sub, sup) boolean function returns true if the given subclass sub is indeed a subclass of the superclass sup.

• The isinstance(obj, Class) boolean function returns true if obj is an instance of class Class or is an instance of a subclass of Class

**10.8 OVERRIDING METHODS**

You can always override your parent class methods. One reason for overriding parent's

methods is because you may want special or different functionality in your subclass.

**10.9 BASE OVERLOADING METHODS**

Following table lists some generic functionality that can be overridden in your own classes−

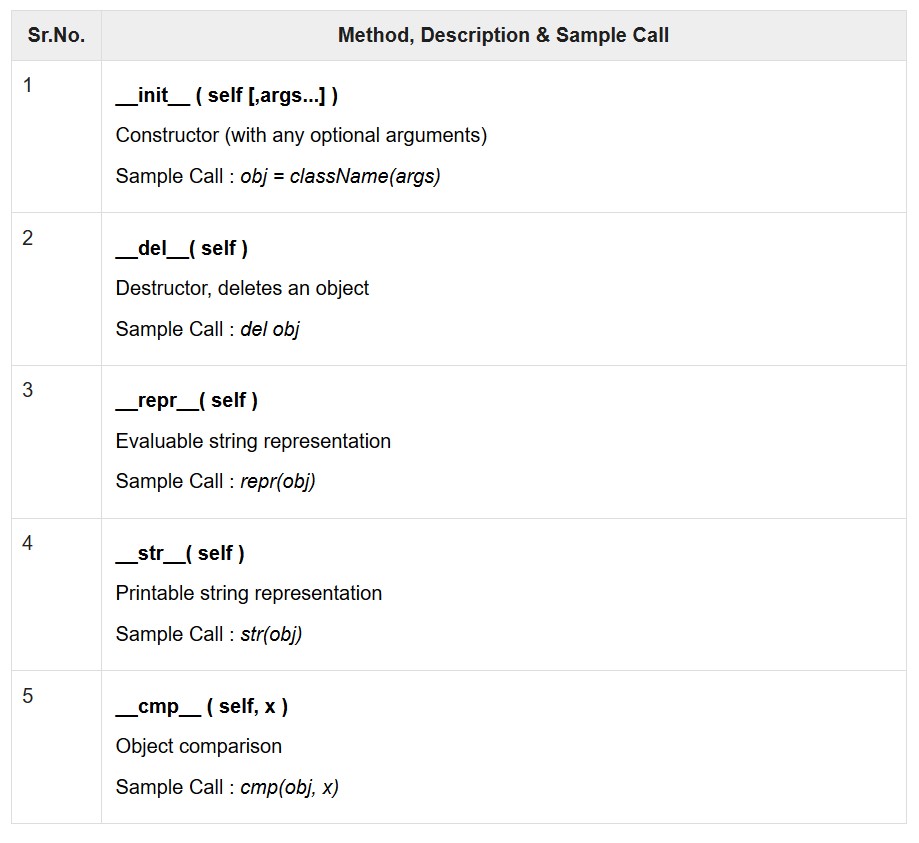


Fig10.3

**10.10 OVERLOADING OPERATORS**

Suppose you have created a Vector class to represent two-dimensional vectors, what happens when you use the plus operator to add them? Most likely Python will yell at you. You could, however, define the \_\_add\_\_ method in your class to perform vector addition and then the plus operator would behave as per expectation

**10.11 DATA HIDING**

An object's attributes may or may not be visible outside the class definition. You need to name attributes with a double underscore prefix, and those attributes then are not be directly visible to outsiders.

Python protects those members by internally changing the name to include the class name. You can access such attributes as object.\_className\_\_attrName.

**CHAPTER-11**

**DEVELOPING GUI WITH PyQT**

**11.1 DESIGNER TOOLS FOR PYQT5**

*11.1.1 FOR WINDOWS:*

1. Open command line window.

2. Run the command: pip3 install pyqt5-tools==5.9.0.1.2

3. Go to your Python installation folder (assuming it is C:\python36)

4. You will find a folder called Lib. You will find designer.exe file in its sub-

directories.

5. Go to the folder C:\Python36\Lib\site-packages\pyqt5-tools

6. You can find designer.exe in this folder. Run that file

*11.1.2 FOR LINUX:*

1. Open terminal.

2. Run the following command: sudo apt-get install qttools5-dev-tools

3. The executable file designer can be found in the following directory:

/usr/lib/i386-linux-gnu/qt5/bin/

**11.2 COMMON WIDGETS**

|  |  |
| --- | --- |
| Application window | A container widget inside which other elements are arranged |
| Button | An area on which a mouse click triggers some operation |
| Textbox | Reads text input from a keyboard |
| Radio button | Allows a user select one of several options |
| Checkbox | Allows a user to select one or more of several options |

**11.3 MAJOR CLASSES IN PYQT**

|  |
| --- |
| QtCore Core non-GUI classes used by other modules |
| QtMultimedia Classes for low-level multimedia  Programming |
| QtNetwork Allows a user select one of several options |
| QtOpenGL OpenGL support classes |
| QtScript Classes for evaluating Qt Scripts |
| QtSql Classes for database integration using SQL |
| QtSvg Classes for displaying the contents of SVG  files |
| QtWebKit Classes for rendering and editing HTML |
| QtXml Classes for handling XML |
| QtWidgets Classes for creating classic desktop-style UIs. |

**11.4 QFORM LAYOUT**

Layout is a convenient way to create a form with two columns and multiple rows.

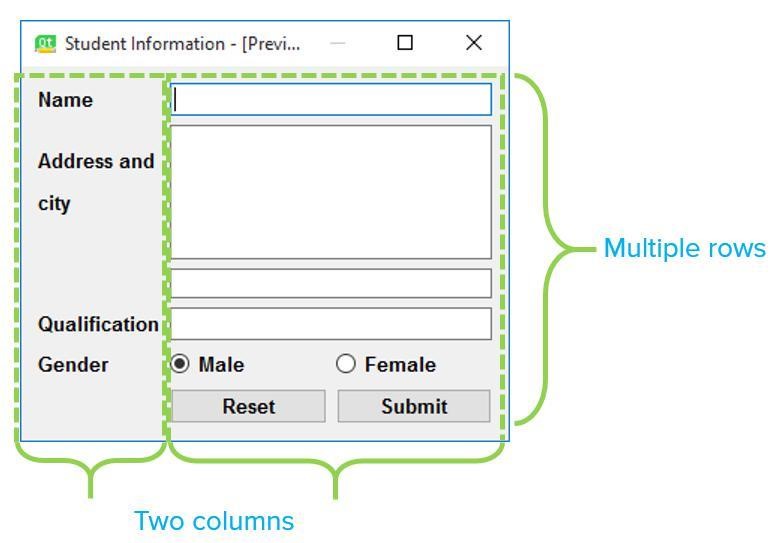


Fig11.1

Each row consists of an input field associated with a label. As a convention, the left column contains the label while the right column contains an input field. TheQFormLayout class has a setWidget() method. Its usage is: QFormLayout.setWidget(rownum, role, widget). The role parameter has two possible values, LabelRole and FieldRole. We shall now design a student information form in the Form Layout. The broad process we will follow is:

• Create a new form.

• Roughly place all the required widgets into the form.

• Use the Layout option, Form layout which will automatically arrange our widgets.

Let's look at the steps:

*Step 1*

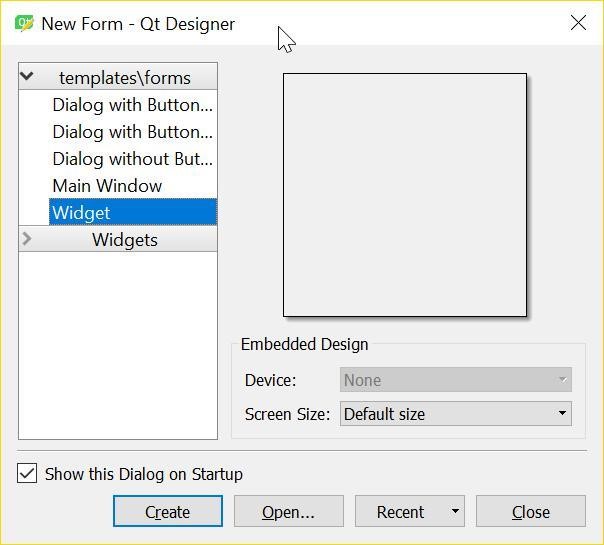


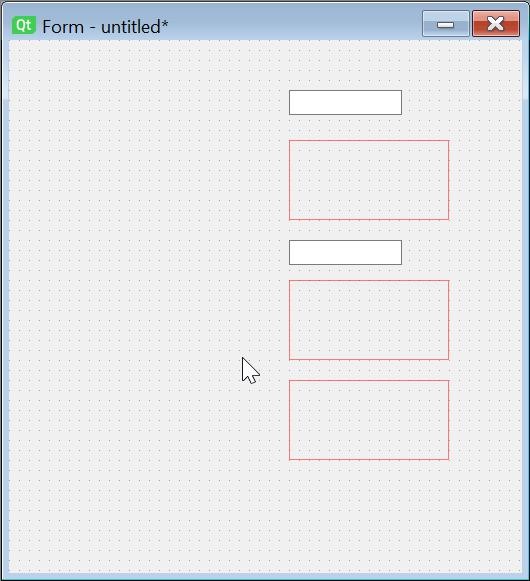
Fig11.2

Start Qt Designer.

Create a new form using the template, Widget.

Use the Layout option, Form layout which will automatically arrange our widgets. .

*Step 2*

 Fig11.3

Let's first place widgets for the right column. You can just drag them and drop them

anywhere.

1. Two Line Edit widgets (Name, Qualification)

2. One vertical layout (For holding Address and City)

3. Two horizontal layout objects (one for holding the gender radio buttons and the other for the Reset and Submit buttons)

*Step 3*

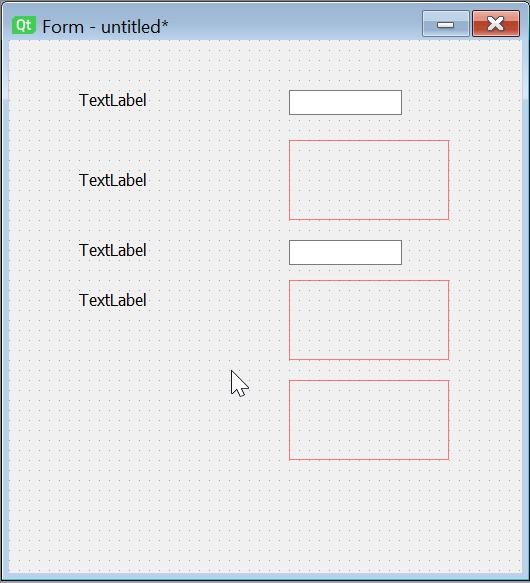


Fig11.4

Let's now place the text labels in the left column.

*Step 4*

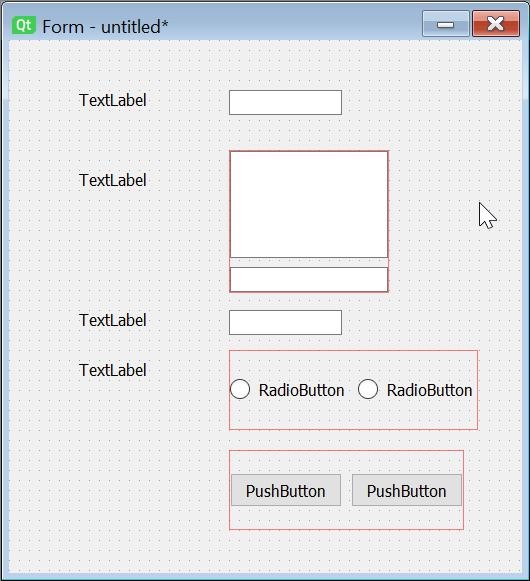


Fig11.5

Now, let's place the required elements into the layouts.

1. First, we place a multi-line text box(address) and a LineEdit (City) widget in the vertical layout.

2. Next, we place two radio buttons (Male, Female) in one horizontal layout.

3. Finally we place two push buttons(Reset, Submit) in the second horizontal

layout.

*Step 5*

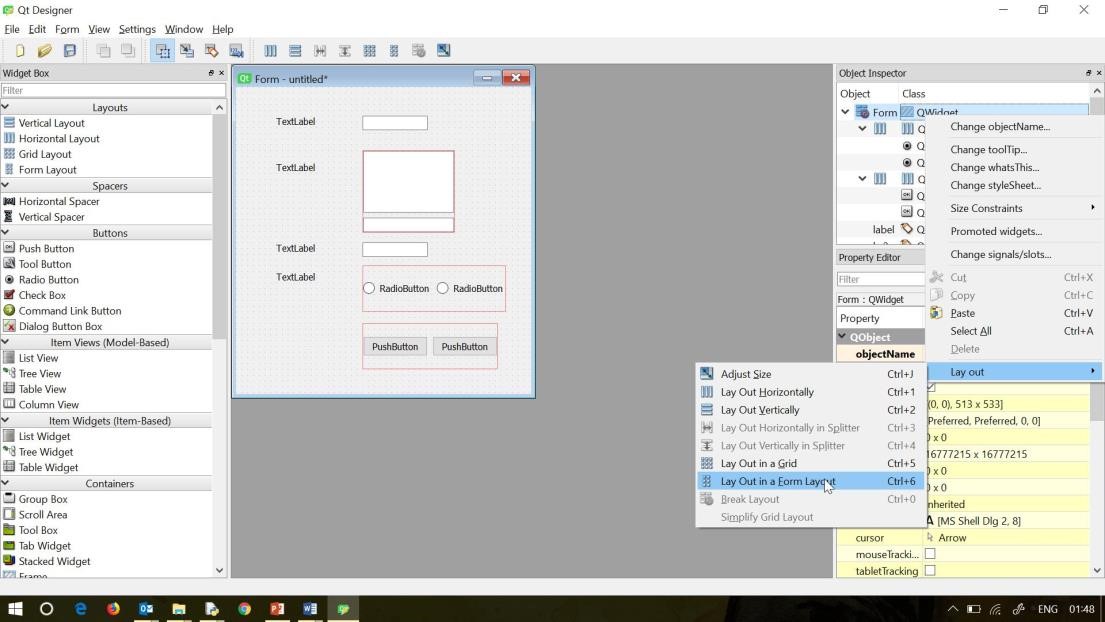


Fig11.6

• Finally, we right-click on QWidget in the Object Inspector Window and select Layout - lay out in form layout. And the widgets are arranged. You can customize the labels and buttons captions before saving the ui file and

**CHAPTER-12**

**APPLICATIONS OF PYTHON IN MODERN WORLD**

**12.1 DATA SCIENCE**

In recent times, Python has shot up in popularity charts mainly because of its Data science libraries. With huge amount of data being ge companies need business insights from this data.

Today Python has become language of choice for data scientists. Python libraries likeNumPy, Pandas and Matplotlib are used extensively in the process of data analysis and collection, processing and cleansing of data nerated by web applications, mobile applications and other devices, sets, applying mathematical algorithms to data and to generate visualizations for the benefit of users. Commercial Python distributions such as Anaconda and Activestate provide all the essential libraries required for data science.

**12.2 MACHINE LEARNING**

This is another glamorous application area where Python developers are getting attracted. Based upon the past data, Python libraries such as Scikit-learn, Tensorflow, and NLTK are widely used for prediction of trends like customer satisfaction, projected values of stocks etc. Some of the real world applications of machine learning are as under:

*Medical Diagnosis*: Machine learning techniques are used for the analysis of the importance of clinical parameters and of their combinations for prognosis, e.g. prediction of disease progression, for the extraction of medical knowledge for outcomes research, for therapy planning and support, and for overall patient management.

Statistical Arbitrage: Machine learning methods are applied to automate trading strategies where user tries to implement a trading algorithm for a set of securities on the basis of quantities such as historical correlations and general economic variables.

*Learning associations:* Machine learning helps the process of developing insights into various associations between products and buying behaviors of customers.

*Basket analysis-* studying the association between the products people buy and

suggesting the associated product to the customer, is a well known phenomenon we see while doing online shopping.

*Prediction:* Current prediction is one of the hottest machine learning algorithms. Businesses are interested in finding out what will be my sales next month / year / Diwali, etc. so that business can take required decision (related to procurement, stocks, etc.) on time.

**12.3 WEB DEVELOPMENT**

Another application area which is becoming increasing popular with Python

developers is web development. Simple to complex web applications can be

developed using easy to use web application frameworks like django, Pyramid, Flask etc. These frameworks are used extensively by various IT companies. Dropbox for example uses django as a backend to store, synchronize local folders.

Most of the web servers today are compatible with WSGI (Web Server Gateway Interface) – a specification for universal interface between Python web frameworks and web servers. All leading web servers such as Apache, IIS, Nginx etc can now host Python web applications. Google’s App Engine hosts web applications built with almost all Python web frameworks.

**12.4 IMAGE PROCESSING**

Face detection and gesture recognition using OpenCV library and Python is

another important application. OpenCV is a C++ library, but has been ported to Python. This library has lot of extremely powerful image processing functions. Facebook’s automatic tag suggestion feature, which used face recognition to suggest people you might want to tag in your photos, is an instance of OpenCV at work. Other instances of OpenCV applications are:

Automated inspection and surveillance.

Robot and driverless car navigation and control.

Medical image analysis.

Video/image search and retrieval.

**12.5 GAME DEVELOPMENT**

Python is a popular choice of game developers. The Pygame library is

extensively used for building games for desktop as well as mobile platforms.

Pygame applications can be installed on Android too.

**12.6 EMBEDDED SYSTEMS AND IOT**

Another important area of Python application is in embedded systems.

Raspberry Pi is a very popular yet a low cost single board computer. It is being

extensively used in automation products, robotics, IoT, and kiosk applications.

Apart from Raspberry Pi, other microcontrollers can also be programmed with

Python. A lightweight version of Python called micropython has been developed especially for microcontrollers. Popular microcontrollers like Arduino are used in many IoT products and programmed with Python. A special micropython compatible controller called Pyboard has also been developed.

**12.7 ANDROID APPS**

Although Android apps are predominantly developed using Android SDK which is more or less like Java, Python can also be used to develop Android apps. Python’s Kivy library has all the functionality required to build a mobile application.

**12.8 AUTOMATED JOBS**

Python is extremely useful and widely used for automating CRON jobs. Certain tasks like backups can be scheduled to be invoked automatically. These tasks are defined in Python scripts and operating system scheduler executes them at predefined times.

Python is embedded as a scripting language in many popular software products. This is similar to VBA used for writing macros in Excel, Powerpoint etc. Similarly Python API is integrated with Maya, PaintShop pro, etc.

**CHAPTER-13**

**MAKING THE CALCULATOR USING PyQT AND PYTHON**

We have used PyQT to make the gui of the calculator and python code to make it work.

I have listed the steps how I made the project.

1. I have selected the Main Window to form the GUI of calculator in the QTDesigner app.

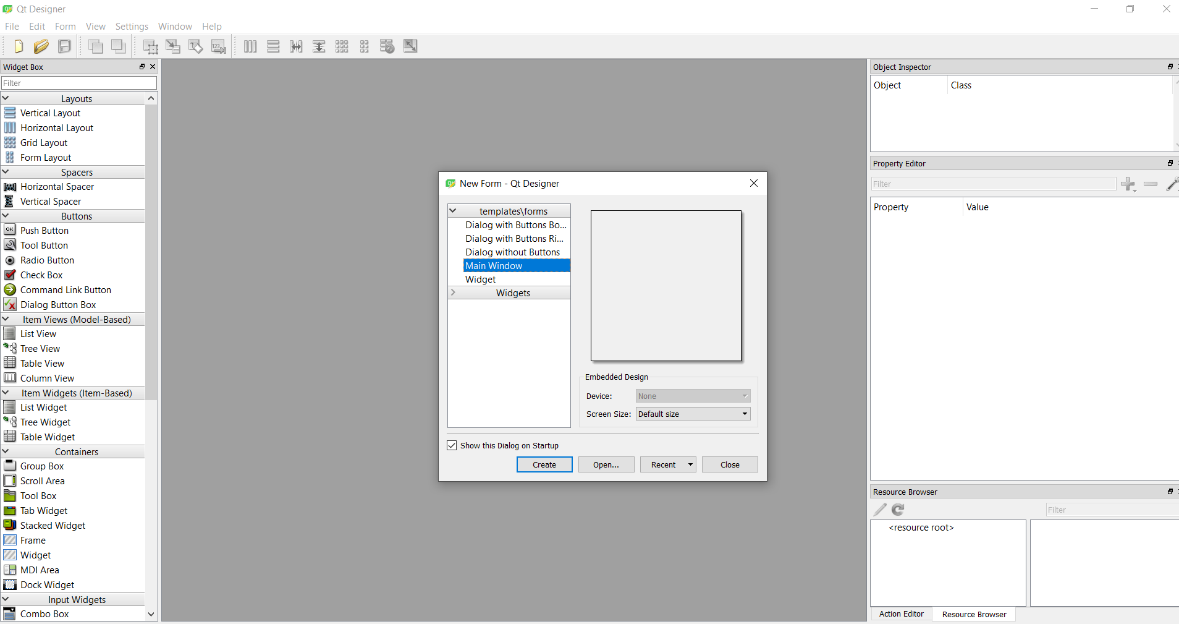


Fig13.1

1. Then I got the following result.

Fig13.2

1. Then I resized the window according to make it look like the outline of a calculator.

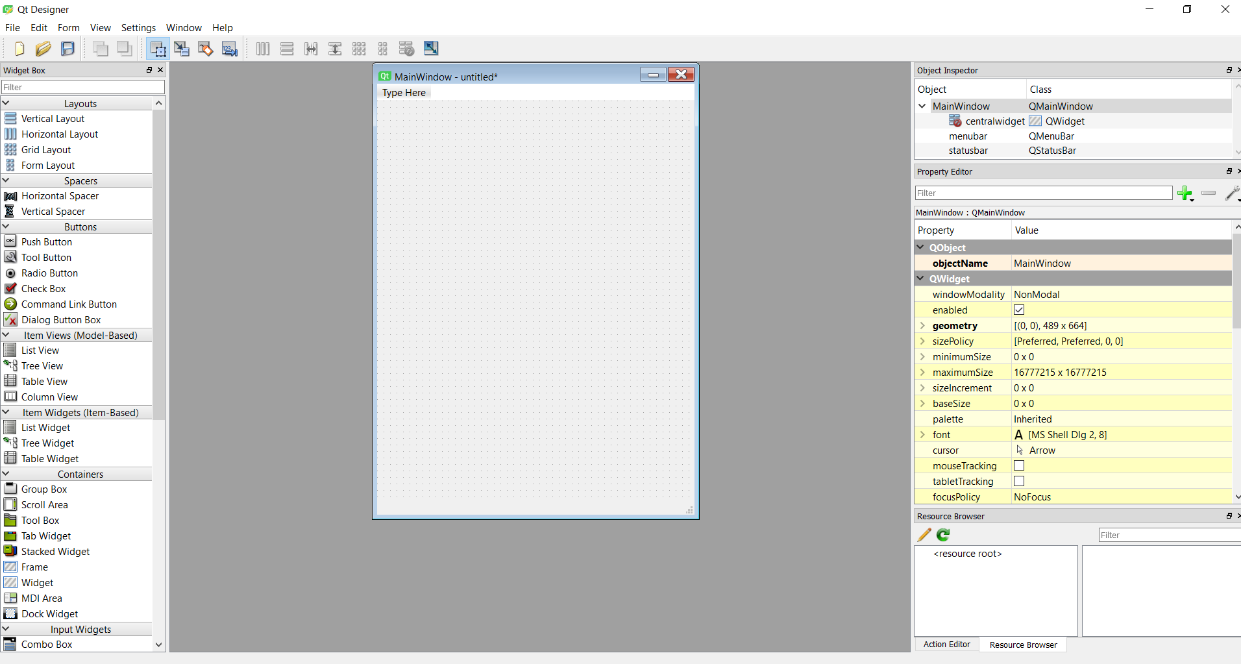


Fig13.3

1. I put the Form Layout and selected Lay Out Vertically to make in inner outline of the calculator.

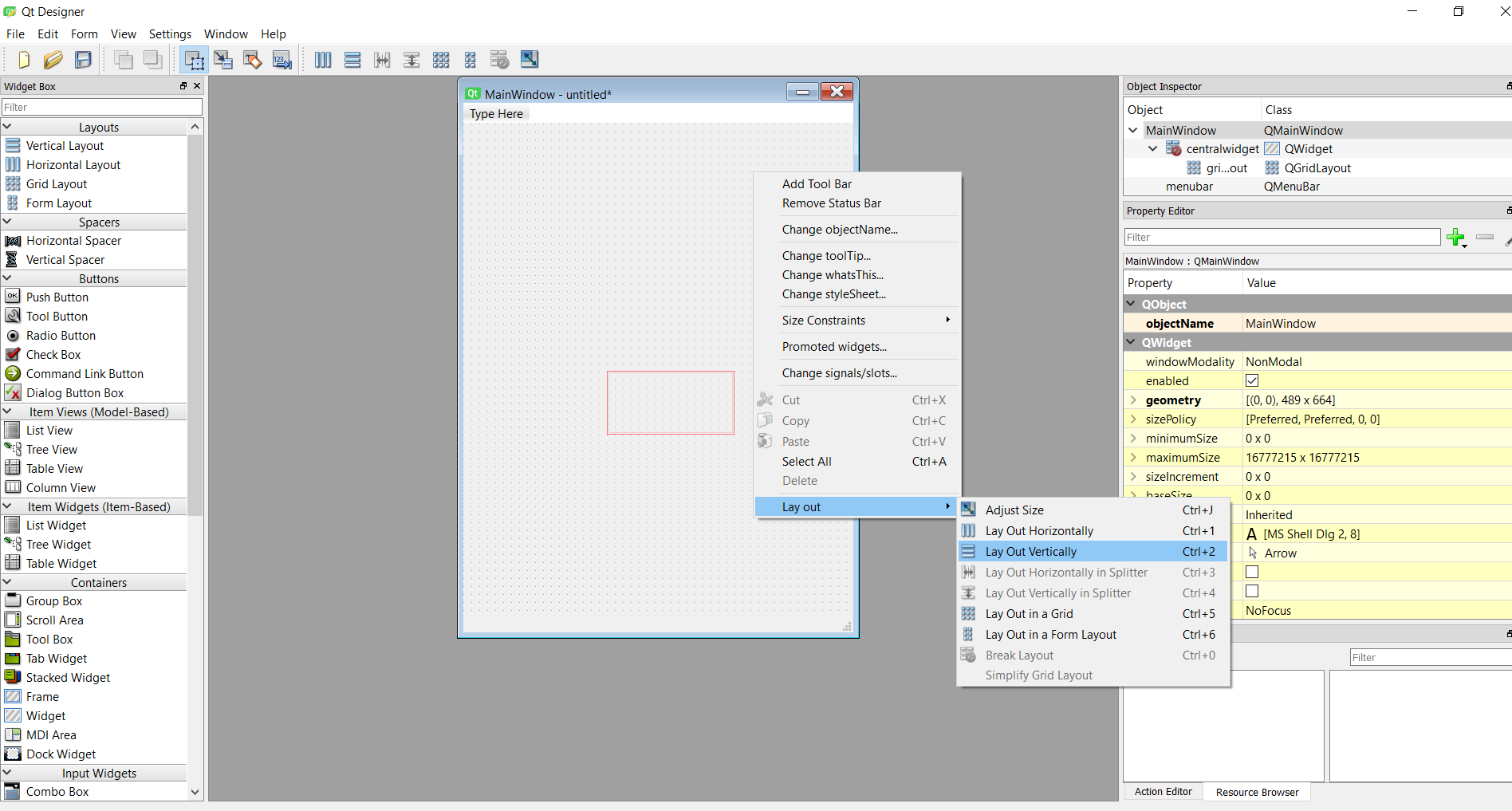


Fig13.4

1. Now I put on the push button and the label as according to a calculator.

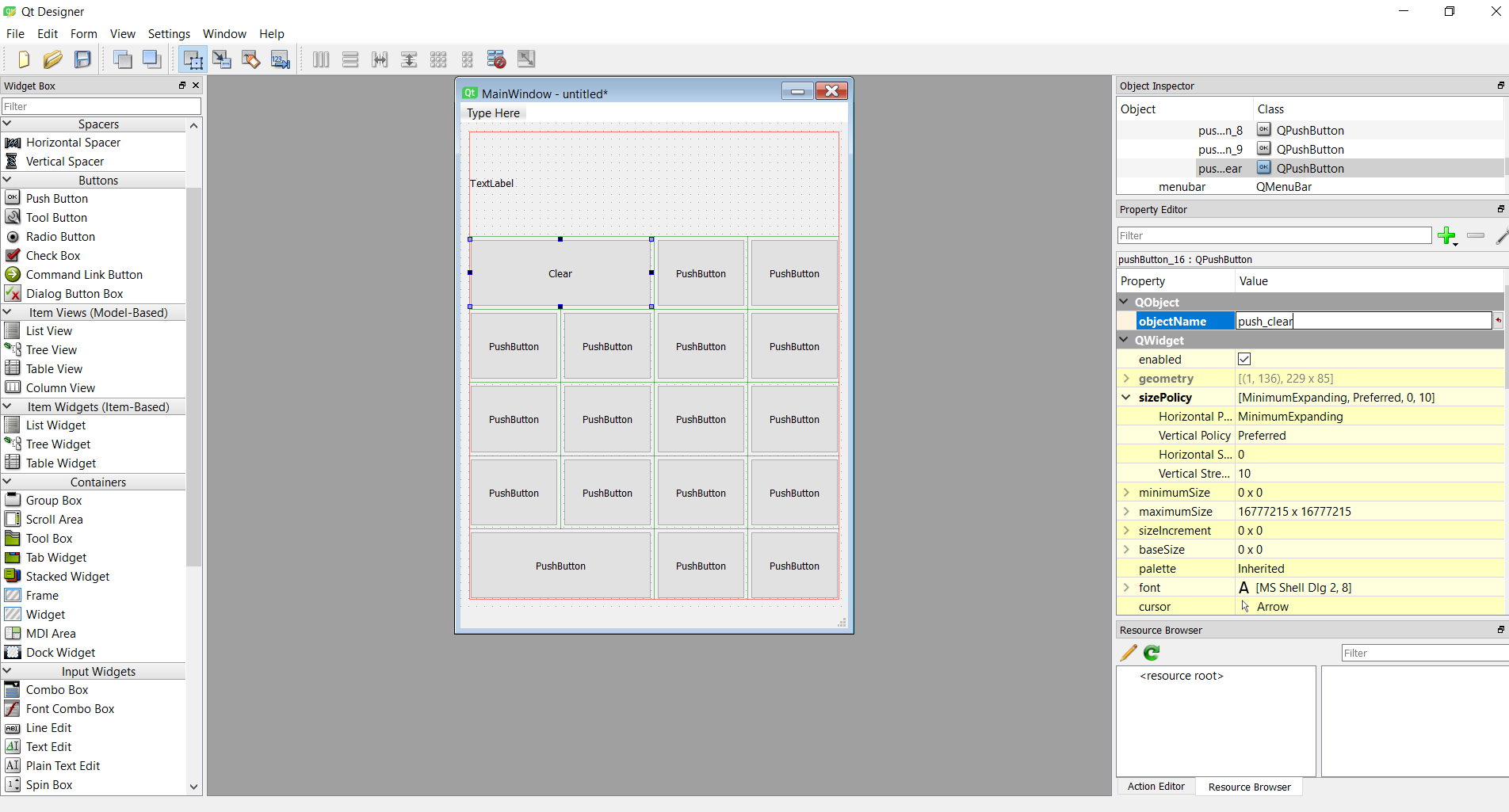


Fig13.5

1. Then I changed the font inside the label tag i.e. where the number will appear when a user uses the calculator.

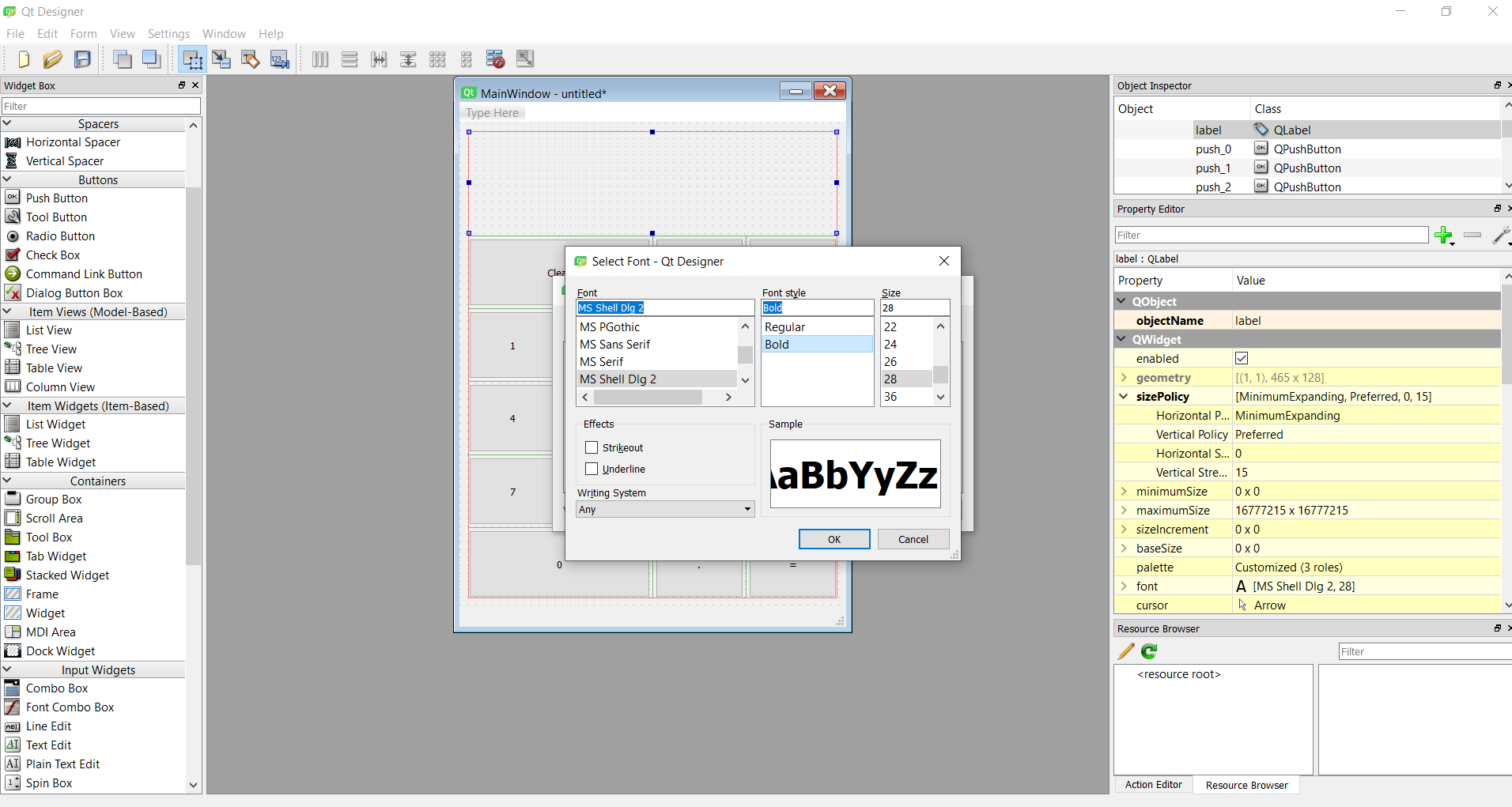


Fig13.6

1. I also selected the colour of border of cells that I liked.

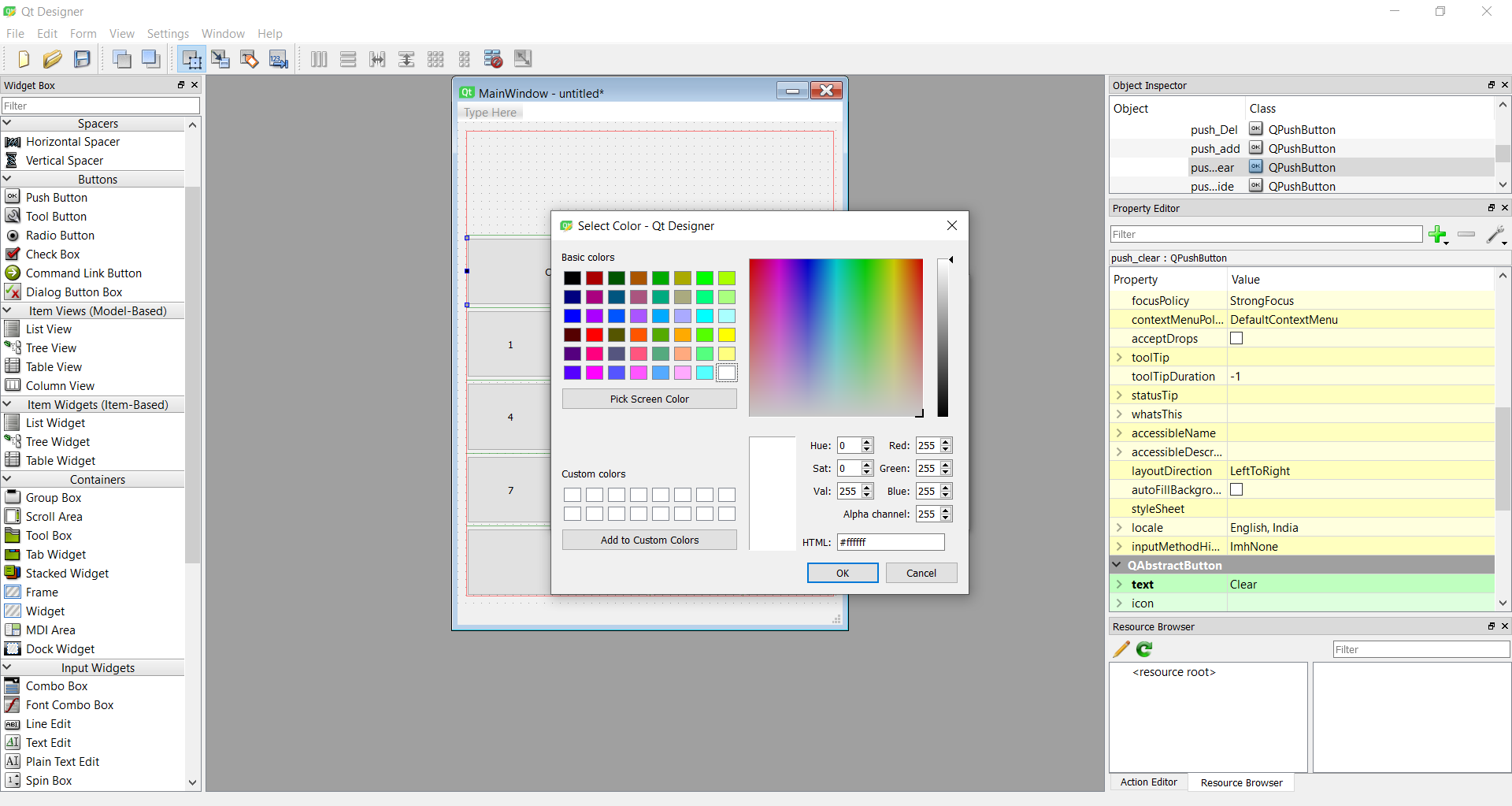


Fig 13.7

1. Then I changed the style of each and every button to make it look more attractive.

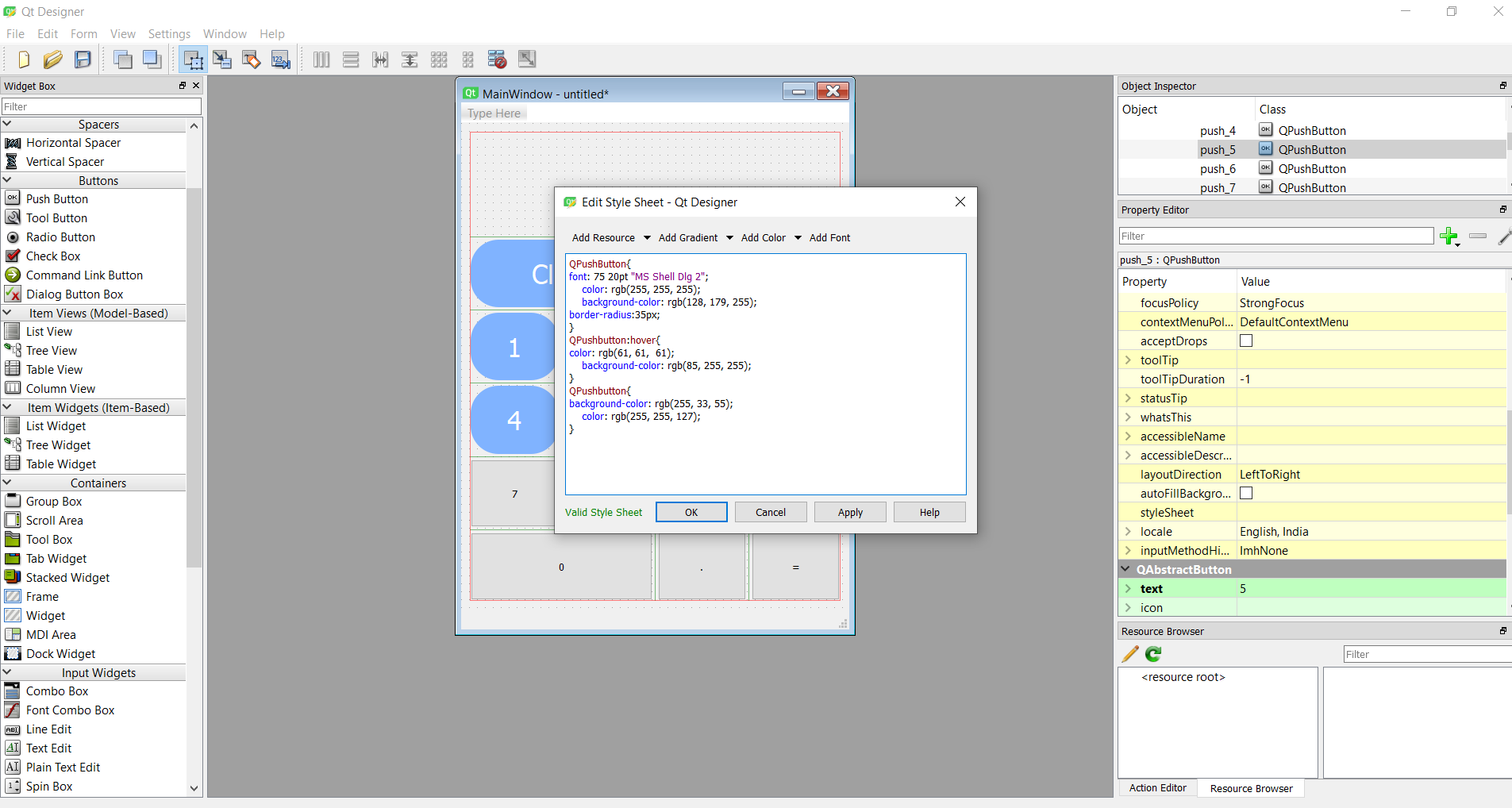


Fig13.8

As you can see I changed the shape, colour for both when cursor will be there on that button or not. Also I added the keyboard shortcut button for each.

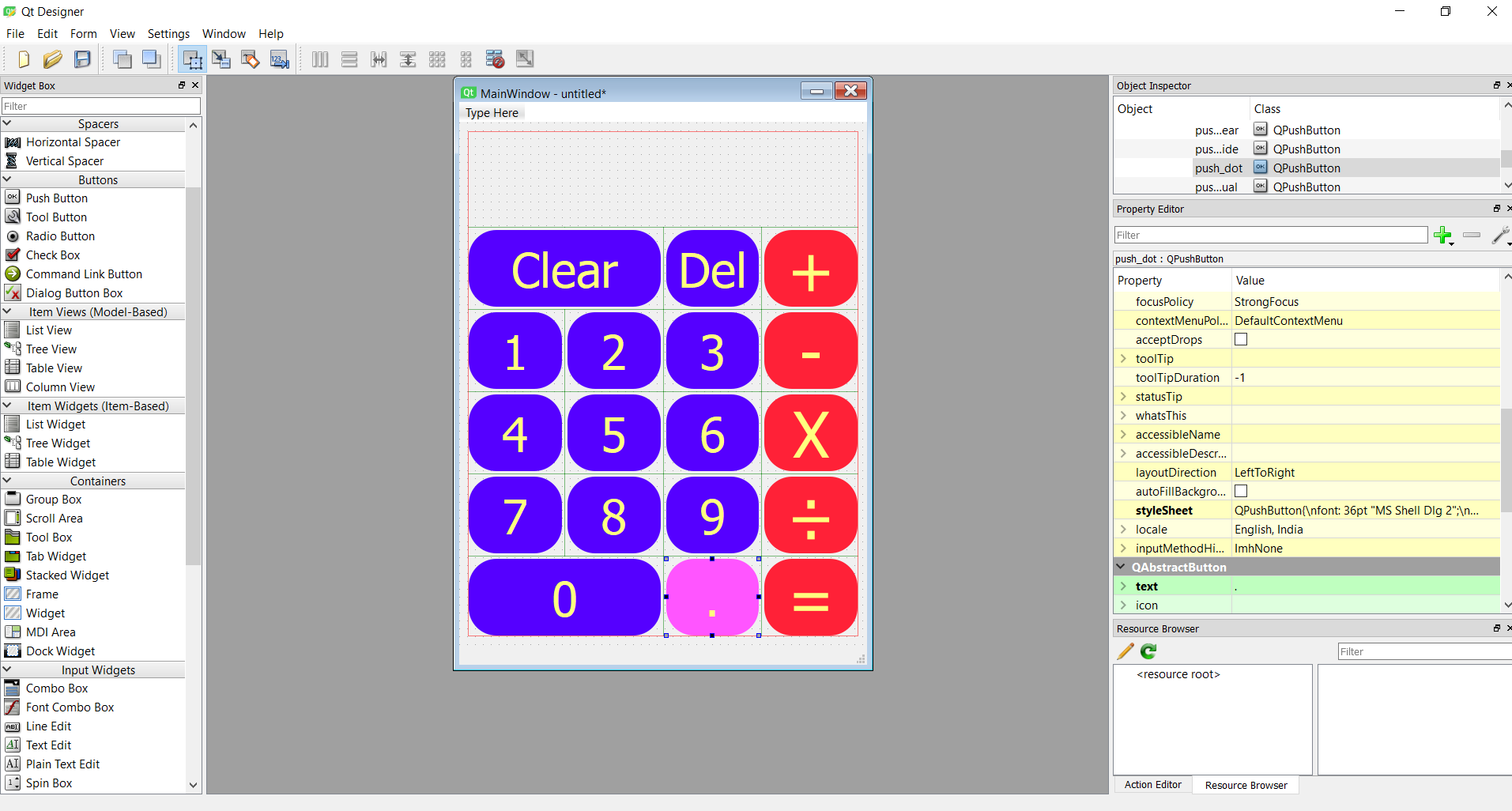
As you can see our gui of calculator is ready but still we have to do some code so as to make it work also.

Fig13.9

For this, we first saved it in .ui format.

Then change it to .py format using command prompt.

Then open the file in any python editor and do the required coding.



Fig13.10



Fig13.11

We can see our calculator is working perfectly.

**BIBLIOGRAPHY**

1. <https://docs.python.org/3/tutorial/index.html>
2. https://www.tutorialspoint.com/python/pyton\_classes\_objects.htm
3. https://trainings.internshala.com/