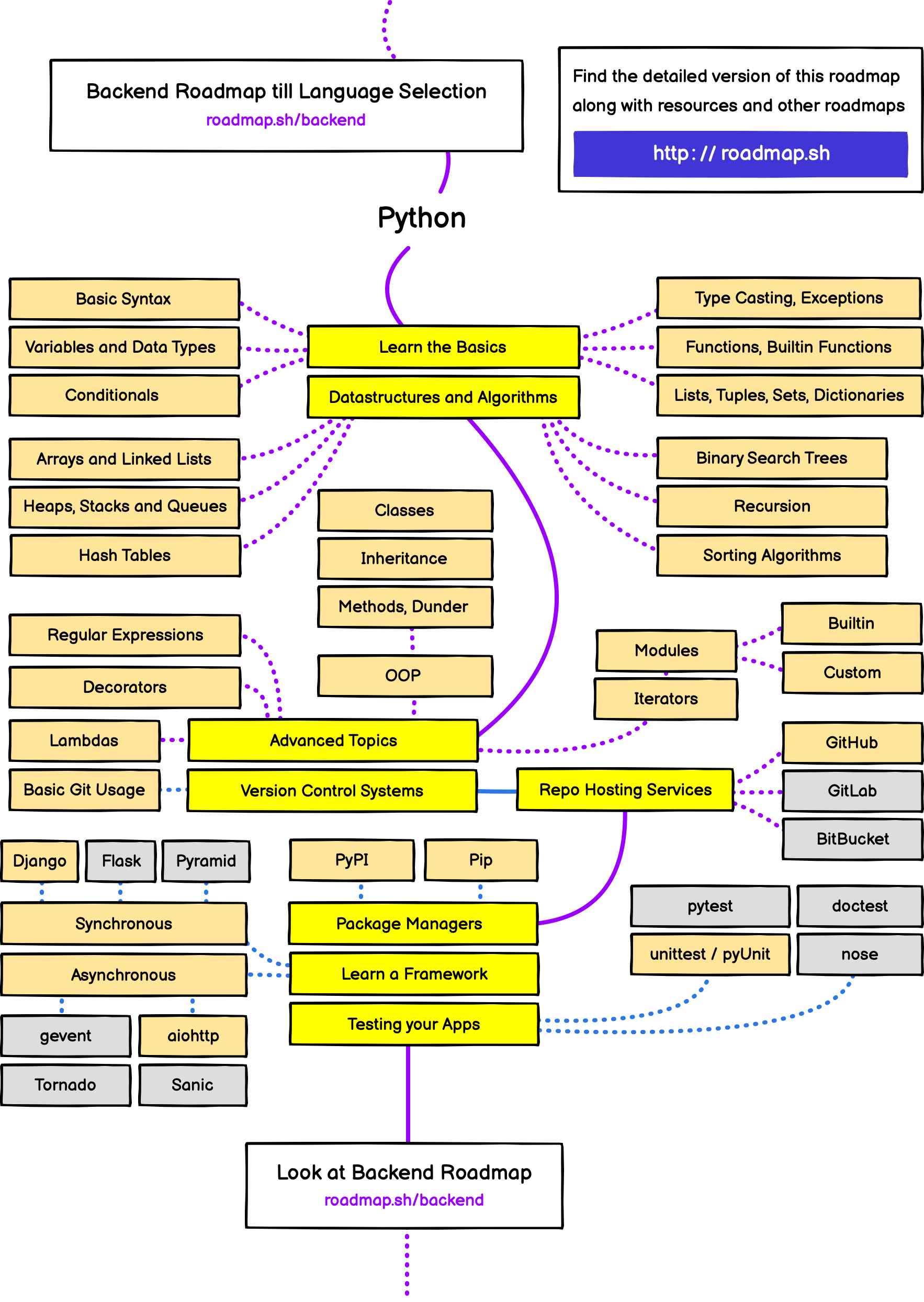
**Python Learning Guide**

****

1. **Learn the basic­­­**

According road map picture basic learning have:-

* 1. Basic Syntax
  2. Variable and Data Type
  3. Conditionals
  4. Type Casting, Exceptions
  5. Functions, Build Functions
  6. Lists, Tuples, Sets, Dictionaries
  7. Basic Syntax

print (‘Hello’)

x=(“Hello”)

print (id(x))

In latest Python version (3.9): **print** keywords must start p is **small letter** and all of the value/string you want to display use with between **open parentheses** and **close parentheses.** print (id(x)) means memory location of **x**

**Quotation in Python:**

Python accepts **single ('), double (") and triple (''' or """) quotes** to denote string literals, as long as the same type of quote starts and ends the string. \* **Used same quote starts and ends**  
Example:

print(‘’’this is sentence one  
this is sentence two  
this is sentence three’’’)

The output is:  
>>>**this is sentence one**>>>**this is sentence two**>>>**this is sentence three**

**Reserved Words**

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

|  |  |  |
| --- | --- | --- |
| And | exec | Not |
| Assert | finally | Or |
| Break | for | Pass |
| Class | from | Print |
| Continue | global | raise |
| Def | if | return |
| Del | import | try |
| Elif | in | while |
| Else | is | with |
| Except | lambda | yield |

**Comments in Python**

A hash sign (**#**) is single line comments and triples single quoted string (**‘’’**) can be used as a multiline comments in start and end.

Example:

‘’’

This is a  
multiline comments  
can use in your program

‘’’

**2. Variables and Data Types**

Variables Types

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored

in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

counter = 100 # An integer assignment

miles = 1000.0 # A floating point

name = "John" # A string

print counter

print miles

print name

This produces the following result −

100

1000.0

John

Python allows you to assign a single value to several variables simultaneously. For example −

a = b = c = 1

You can also assign multiple objects to multiple variables. For example −

a,b,c = 1,2,"john"

Standard Data Types

Python has five standard data types −

* Numbers
* String
* List
* Tuple
* Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example –

var1 = 1

var2 = 10

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is −

del var1[,var2[,var3[....,varN]]]]

You can delete a single object or multiple objects by using the del statement. For example −

del var

del var\_a, var\_b

Python supports four different numerical types −

int (signed integers)

long (long integers, they can also be represented in octal and hexadecimal)

float (floating point real values)

complex (complex numbers)

**Examples**

Here are some examples of numbers –

|  |  |  |  |
| --- | --- | --- | --- |
| **int** | **Long** | **float** | **complex** |
| 10 | 51924361L | 0.0 | 3.14j |
| 100 | -0x19323L | 15.20 | 45.j |
| -786 | 0122L | -21.9 | 9.322e-36j |
| 080 | 0xDEFABCECBDAECBFBAEl | 32.3+e18 | .876j |
| -0490 | 535633629843L | -90. | -.6545+0J |
| -0x260 | -052318172735L | -32.54e100 | 3e+26J |
| 0x69 | -4721885298529L | 70.2-E12 | 4.53e-7j |

* Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
* A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit.

## **Python Strings**

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (\*) is the repetition operator. For example –

#!/usr/bin/python

str = 'Hello World!'

print (str) # Prints complete string

print (str[0]) # Prints first character of the string

print (str[2:5]) # Prints characters starting from 3rd to 5th

print (str[2:]) # Prints string starting from 3rd character

print (str \* 2) # Prints string two times

print (str + "TEST") # Prints concatenated string

This will produce the following result −

Hello World!

H

llo

llo World!

Hello World!Hello World!

Hello World!TEST

## **Python Lists**

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (\*) is the repetition operator. For example −

#!/usr/bin/python

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tinylist = [123, 'john']

print (list) # Prints complete list

print (list[0]) # Prints first element of the list

print (list[1:3]) # Prints elements starting from 2nd till 3rd ၁ ကနေစမယ်

အရေအတွက် အားလုံးရဲ့ ၃ ခုမြောက်ထိ ပြမယ်

print (list[2:]) # Prints elements starting from 3rd element

print (tinylist \* 2) # Prints list two times

print (list + tinylist) # Prints concatenated lists

This produce the following result −

['abcd', 786, 2.23, 'john', 70.2]

abcd

[786, 2.23]

[2.23, 'john', 70.2]

[123, 'john', 123, 'john']

['abcd', 786, 2.23, 'john', 70.2, 123, 'john']

## **Python Tuples**

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated. Tuples can be thought of as **read-only** lists. For example −

#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

tinytuple = (123, 'john')

print tuple # Prints the complete tuple

print tuple[0] # Prints first element of the tuple

print tuple[1:3] # Prints elements of the tuple starting from 2nd till 3rd

print tuple[2:] # Prints elements of the tuple starting from 3rd element

print tinytuple \* 2 # Prints the contents of the tuple twice

print tuple + tinytuple # Prints concatenated tuples

This produce the following result −

('abcd', 786, 2.23, 'john', 70.2)

abcd

(786, 2.23)

(2.23, 'john', 70.2)

(123, 'john', 123, 'john')

('abcd', 786, 2.23, 'john', 70.2, 123, 'john')

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists −

#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tuple[2] = 1000 # Invalid syntax with tuple

list[2] = 1000 # Valid syntax with list

sample of Tuple and List usage code:

|  |  |
| --- | --- |
| tinylist =['abcd',786,28,'Aye Co',82]  print (tinylist [1:3])  print(tinylist \*2)  print('list location is : ' + str(id(tinylist)))  print(id(tinylist))  name=input('Enter Name List with comma separate')  nameList=name.split(",")  print(nameList)  print(type(nameList))  print(nameList[2:])  nameTuple=tuple(nameList)  print(type(tuple(nameList)))  print(type(list(nameList))) | \* tinylist ကို တခါတည်း Variable ထည့် \* print out ထုတ်  \*list ကို ၂ ကြောင်း print ထုတ်  \*list ရဲ့ memory location ကို ပြရန်  \* list ရဲ့ id ကို ပြရန်  \*comma separate နဲ့ string ၁ ခုကို input တောင်း  \*nameList ဆိုတဲ့ list ထဲကို comma နဲ့ ခွဲ ပြီး value ထည့်  \*nameList ကို print out ထုတ်  \* nameList ရဲ့ variable အမျိုးအစားကို print out ထုတ်  \*nameList ရဲ့ List ၂ခုမြောက်က စ၍ ကျန်တာ အကုန်ပြ  \*nameList ကို list မှ tuple သို့ format ပြောင်း  \*nameList ကို tuple format သို့ ပြောင်းပြီး variable type ကို print out ထုတ်  \*nameList ကို list format သို့ ပြောင်းပြီး list type ကို ပြ |

\*\* Tuple သုံးလျှင် လက်သည်းကွင်း။ List သုံးလျှင် ထောင့်ကွင်း။ lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type. Thus, List is different data type and can read, write and edit. The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated. Tuples can be thought of as read-only lists.

## **Python Dictionary**

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example −

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print dict['one'] # Prints value for 'one' key

print dict[2] # Prints value for 2 key

print tinydict # Prints complete dictionary

print tinydict.keys() # Prints all the keys

print tinydict.values() # Prints all the values

This produce the following result −

This is one

This is two

{'dept': 'sales', 'code': 6734, 'name': 'john'}

['dept', 'code', 'name']

['sales', 6734, 'john']

Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

\* **Dictionaries are changeable**, meaning that we can change, **add or remove** items after the dictionary has been created but **duplicates not allowed. If dictionary key is duplicate/same, last key’s value is overwrite to the old one. If you want to use duplicate, create a nested dictionary format. Following code format is nested dictionary style. If we need to record student data at that time, we can used Nested Dictionary. Check following code example.**

|  |
| --- |
| people={  1:{'name':'John',’grade’:’B’,'age':'28','sex':'Male'},  2:{'name':'Marie', ’grade’:’B’,'age':'33','sex':'Female'},  3:{'name':'Tuple', ’grade’:’B’,'age':'22','sex':'Bisex'},  4:{'name':'List', ’grade’:’B’,'age':'220','sex':'Unknown'}  } |

If you want to retrieve value only from Nested Dictionary, check following code:

|  |
| --- |
| thisList=['apple','banana','orange'] #list create  for p\_id,p\_info in people.items(): #Looping with dictionary keys    for key in p\_info: #Looping again with values change as second key  thisList.append(p\_info[key]) #insert retrieve value to thisList array  print(thisList) |