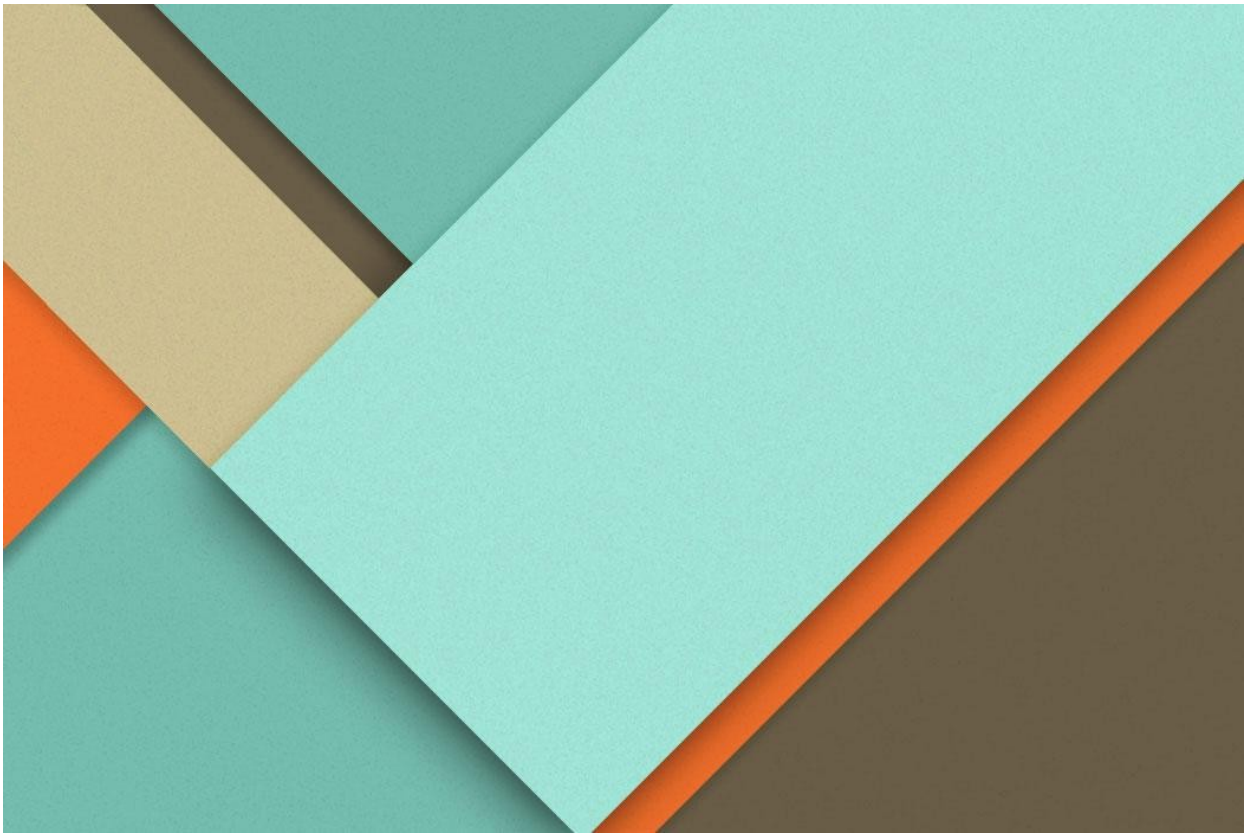


Surge Virtual Python Assistant

January 2019

—

Computer Science Project File

Parth Kumar
Madhav Krishnan
Rishi Moorthy
XI-B

Introduction

The Virtual AI Companion is designed to show you around Python and make the learning experience much more fun by keeping you entertained. This assistant can guide you throw various Python topics, statements, functions and commands. It also can show you the time, give you suggestions run basic games etc. The Python Assistant was designed to be user friendly and can run with the simplest of inputs.

Features of the AI companion include -

1. Help with Python
2. Magic 8Ball Commands
3. Google Something
4. Load up random facts
5. Suggest Fun Websites
6. Run Tic Tac Toe
7. Display the date
8. Display the calendar
9. Flip a coin

Most features in the code are randomized to make the AI seem more realistic, the text flows in slowly too which gives it a more personified view. The interface is friendly and extremely accessible. Python veterans would be able to quickly grasp concepts with the help of this AI.

Detailed Description and Technical Description

The Python Code Consists of the following :-

1) Introduction - The AI introduces itself with simple Print and Input commands. It uses the time.sleep and print_slow functions to display the text slowly. The duration between each lexicon and statement is mentioned in the code (0.003 and 1 second) It proposed random text by using the random.choice function

2) Flip A Coin - If a user inputs “flip” a slight animation of a coin flip is shown using the shape to text editor available online. The image is just text with special symbols which is slowed down by the time.sleep function. The result of the coin flip is randomized and “T” or “H” appears which represents heads or tails. Each part is displayed after 0.5 seconds.

3) Magic8Ball - The Magic8ball was made using while loops, similar to the coin flip, a shape to text editor was used to display the Magic8Ball. The Magic8ball has 13 randomized replies based on which integer is picked randomly from 1-13 inside the while loop.

4) Help With Python - The AI helps with common python commands. It can guide you through `Numbers,String,Lists,Looping,Tuples and Dictionaries`. It works by simple if/else statements. The user may enter what he/she wants to learn about. It gives you a description of each as they are inputted. This is similar to the Python help feature but more user friendly and advanced.

5) Calendar - The calendar was imported through the help of the import function. It takes input from the user which includes the month and date and displays it in the form of a calendar. Uses simple elif statements with imported commands.

6) Time - Similar to the calendar, it allows the user to set the time and the time is then displayed. The time function has also been imported.

7) Website- The website recommender recommends a random website from a list of websites picked by us. The website link is displayed with a small description of the website. This was also made using if elif statements along with random functions.

CODE

```
import time
import random
import sys
usertopop=""
def print_slow(str):
    for letter in str:
        sys.stdout.write(letter)
        sys.stdout.flush()
        time.sleep(0.03)
userop=""
print_slow("Hi! :)")
print("")
time.sleep(1)
print_slow("Im Surge Python assistant")
print("")
time.sleep(1)
while userop!="quit":
    print_slow("I can do things like")
    print("")
```

splash=["Flip a coin", "Summon Magic 8 Ball", "Help you with Python code", "Provide a google search link", "Play Tic tac toe", "Check this months Calendar","Tell you the time" ,"Recommend a website"]

splash1=random.choice([0,1])

splash2=random.choice([2,3])

splash3=random.choice([4,5])

splash4=random.choice([6,7])

time.sleep(0.7)

print_slow(splash[splash1])

print("")

time.sleep(0.7)

print_slow(splash[splash2])

print("")

time.sleep(0.7)

print_slow(splash[splash3])

print("")

time.sleep(0.7)

print_slow(splash[splash4])

print("")

print_slow("What do you want me to do")

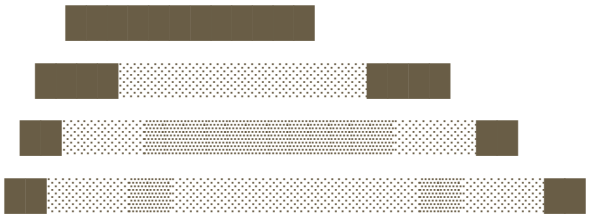
print("")

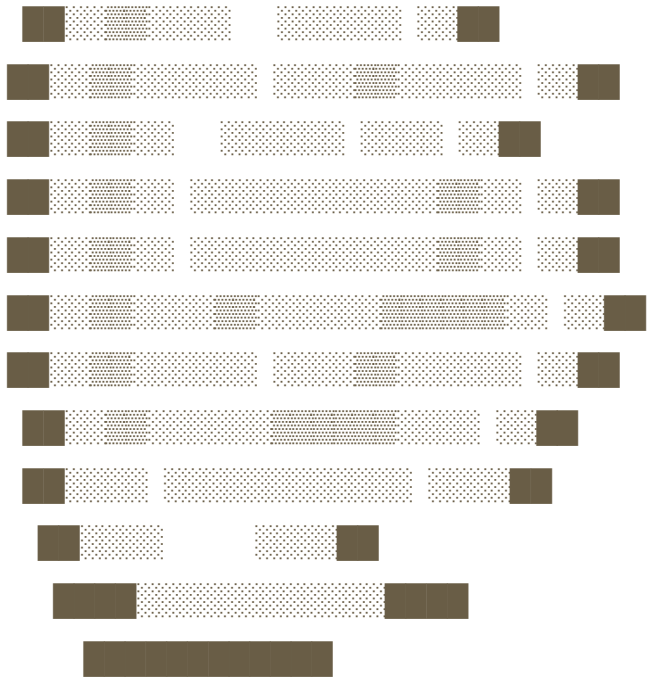
time.sleep(0.7)

userop=input("")

if "flip" in userop or "Flip" in userop or "Coin" in userop or "coin" in userop is True:

print("""flipping coin

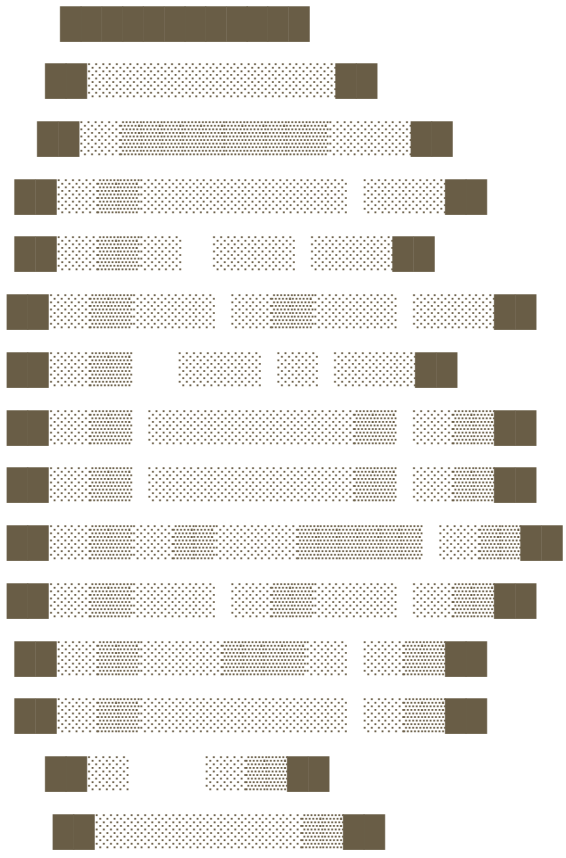




"""")

time.sleep(0.5)

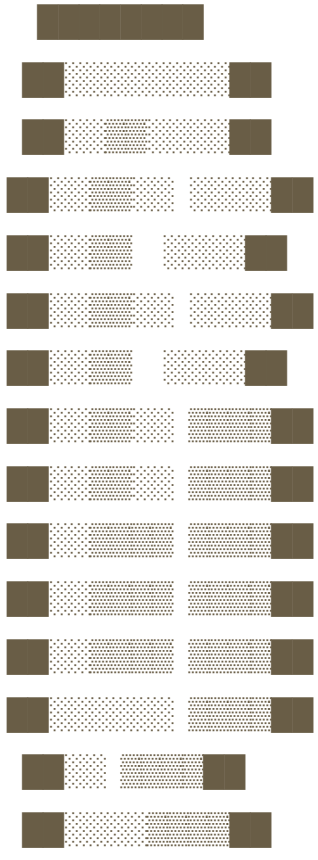
print("""




"""")

time.sleep(0.5)

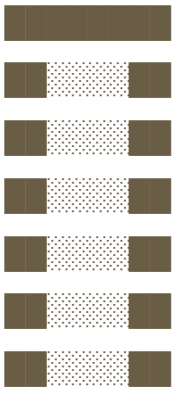
print("""

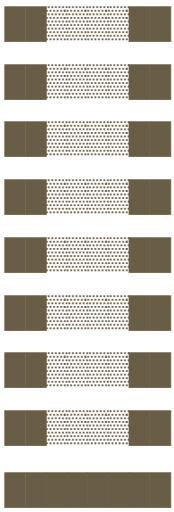



"""")

time.sleep(0.5)

print("""

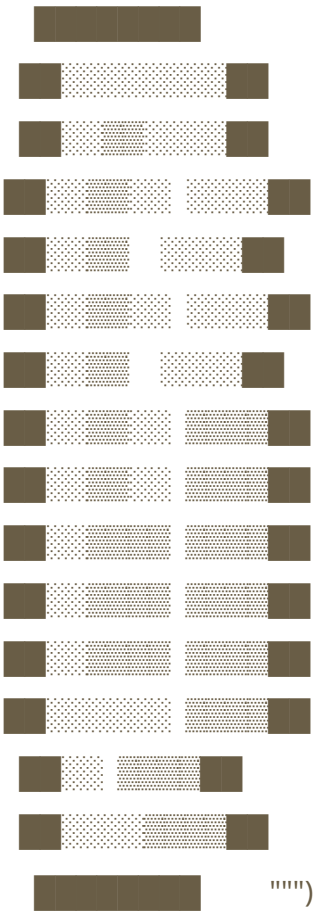




"""")

time.sleep(0.5)

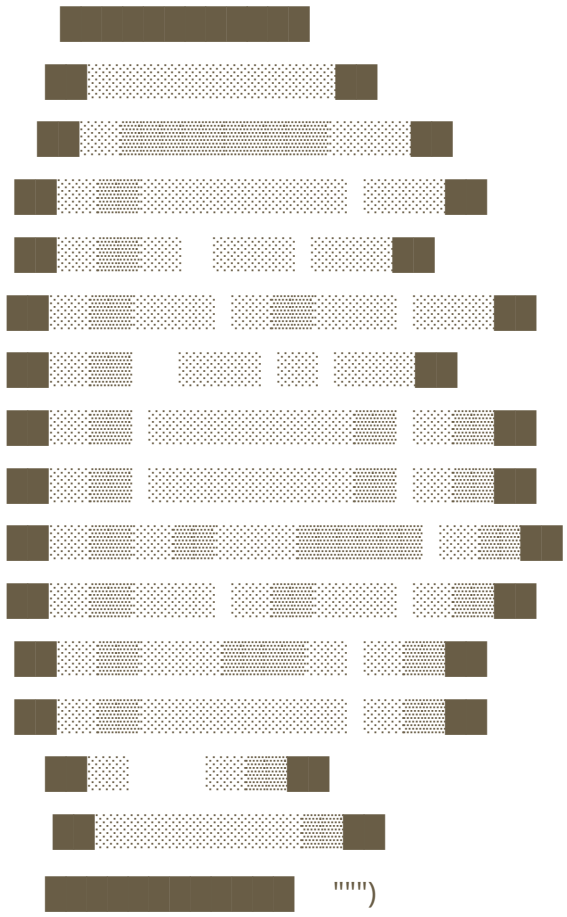
print("""



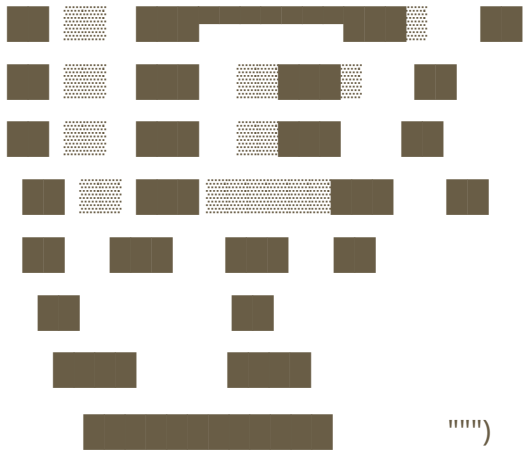
""")

time.sleep(0.5)

print("""



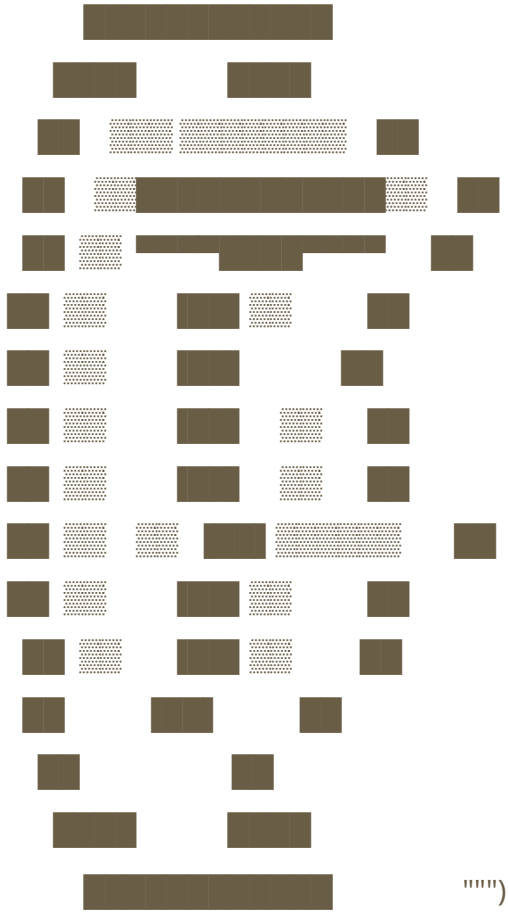
```
    """)

    time.sleep(0.5)
    C_val=random.randint(0,2)
    if C_val==1:
        print("""
```



""")

else :

print("""



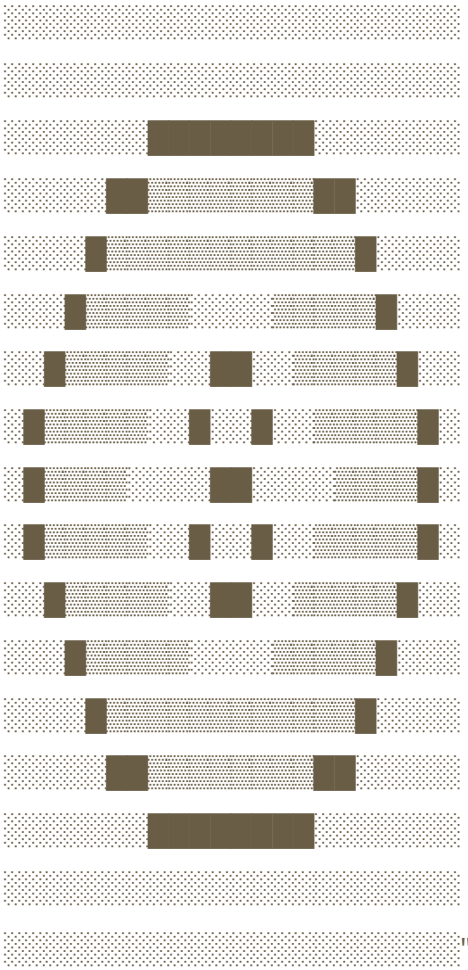
"""")

MAGIC 8 BALL CODE

elif "Magic" in userop or "magic" in userop or "8" in userop or "8" in userop or "ball" in userop or "Ball" in userop is True:

q=""

```
x=0
while q!="quit":
    a=0
    print_slow("""This is the magic 8 ball ask a question or type quit to quit""")
    print("")
    q=input("Enter Question here : ")
    x=len(q)
    while x>=14:
        x=x-13
    while a!=13:
        print("""Shaking the ball
```



```
""")
        a+=1
```

```
        if x==1:
            print("My sources say no.")
        elif x==2:
            print("Yes – definitely")
        elif x==3:
            print("Without a doubt.")
        elif x==4:
            print("It is decidedly so.")
        elif x==5:
            print("Signs point to yes.")
        elif x==6:
            print("Yes.")
        elif x==7:
            print("Most likely.")
        elif x==8:
            print("My reply is no.")
        elif x==9:
            print("Definitly not.")
        elif x==10:
            print("Outlook good.")
        elif x==11:
            print("It is certain.")
        elif x==12:
            print("As I see it, yes.")
        elif x==13:
            print("Outlook not so good.")
#HELP WITH PY
```

elif "Python" in userop or "python" in userop or "Help" in userop or "help" in userop is True:

while usertopop!="quit":

print_slow("Surge can help you with py topics like...")

print("")

print_slow("""Numbers,String,Lists,Looping,Tuples,Dictionaries,if statements""")

print("")

print_slow("""What would you like to learn today?""")

print("")

usertopop=input("")

if "numbers" in usertopop or "Numbers" in usertopop is True:

print("""

Numbers

The interpreter acts as a simple calculator: you can type an expression at it and it will write the value. Expression syntax is straightforward: the operators +, -, * and / work just like in most other languages (for example, Pascal or C); parentheses (()) can be used for grouping. For example:

>>>

>>> 2 + 2

4

>>> 50 - 5*6

20

>>> (50 - 5*6) / 4

5.0

>>> 8 / 5 # division always returns a floating point number

1.6

The integer numbers (e.g. 2, 4, 20) have type int, the ones with a fractional part (e.g. 5.0, 1.6) have type float. We will see more about numeric types later in the tutorial.

Division (/) always returns a float. To do floor division and get an integer result (discarding any fractional result) you can use the // operator; to calculate the remainder you can use %:

```
>>>
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3  # floor division discards the fractional part
5
>>> 17 % 3  # the % operator returns the remainder of the division
2
>>> 5 * 3 + 2  # result * divisor + remainder
17
```

With Python, it is possible to use the ** operator to calculate powers 1:

```
>>>
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

The equal sign (=) is used to assign a value to a variable. Afterwards, no result is displayed before the next interactive prompt:

```
>>>
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

If a variable is not “defined” (assigned a value), trying to use it will give you an error:

```
>>>
```

>>> n # try to access an undefined variable

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

NameError: name 'n' is not defined

There is full support for floating point; operators with mixed type operands convert the integer operand to floating point:

>>>

>>> 4 * 3.75 - 1

14.0

In interactive mode, the last printed expression is assigned to the variable _. This means that when you are using Python as a desk calculator, it is somewhat easier to continue calculations, for example:

>>>

>>> tax = 12.5 / 100

>>> price = 100.50

>>> price * tax

12.5625

>>> price + _

113.0625

>>> round(_, 2)

113.06

This variable should be treated as read-only by the user. Don't explicitly assign a value to it — you would create an independent local variable with the same name masking the built-in variable with its magic behavior.

In addition to int and float, Python supports other types of numbers, such as Decimal and Fraction. Python also has built-in support for complex numbers, and uses the j or J suffix to indicate the imaginary part (e.g. 3+5j)."""")

#STRING

    elif "String" in usertopop or "string" in usertopop is True:

print("""Strings

Besides numbers, Python can also manipulate strings, which can be expressed in several ways. They can be enclosed in single quotes ('...') or double quotes ("...") with the same result 2. \ can be used to escape quotes:

>>>

>>> 'spam eggs' # single quotes

'spam eggs'

>>> 'doesn\'t' # use \' to escape the single quote...

"doesn't"

>>> "doesn't" # ...or use double quotes instead

"doesn't"

>>> '"Yes," they said.'

'"Yes," they said.'

>>> "\"Yes,\" they said."

'"Yes," they said.'

>>> '"Isn\'t," they said.'

'"Isn\'t," they said.'

In the interactive interpreter, the output string is enclosed in quotes and special characters are escaped with backslashes. While this might sometimes look different from the input (the enclosing quotes could change), the two strings are equivalent. The string is enclosed in double quotes if the string contains a single quote and no double quotes, otherwise it is enclosed in single quotes. The print() function produces a more readable output, by omitting the enclosing quotes and by printing escaped and special characters:

>>>

>>> '"Isn\'t," they said.'

'"Isn\'t," they said.'

>>> print('"Isn\'t," they said.')

"Isn't," they said.

>>> s = 'First line.\nSecond line.' # \n means newline

```
>>> s  # without print(), \n is included in the output
'First line.\nSecond line.'
>>> print(s)  # with print(), \n produces a new line
First line.
Second line.
```

If you don't want characters prefaced by \ to be interpreted as special characters, you can use raw strings by adding an r before the first quote:

```
>>>
>>> print('C:\some\name')  # here \n means newline!
C:\some
ame
>>> print(r'C:\some\name')  # note the r before the quote
C:\some\name
```

String literals can span multiple lines. One way is using triple-quotes: ("""...""") or '''...'''. End of lines are automatically included in the string, but it's possible to prevent this by adding a \ at the end of the line. The following example:

```
print("""\
Usage: thingy [OPTIONS]
    -h                  Display this usage message
    -H hostname         Hostname to connect to
""")
```

produces the following output (note that the initial newline is not included):

```
Usage: thingy [OPTIONS]
    -h                  Display this usage message
    -H hostname         Hostname to connect to
```

Strings can be concatenated (glued together) with the + operator, and repeated with *:

```
>>>
>>> # 3 times 'un', followed by 'ium'
```

```
>>> 3 * 'un' + 'ium'
'ununununium'
```

Two or more string literals (i.e. the ones enclosed between quotes) next to each other are automatically concatenated.

```
>>>
>>> 'Py' 'thon'
'Python'
```

This feature is particularly useful when you want to break long strings:

```
>>>
>>> text = ('Put several strings within parentheses '
...         'to have them joined together.')
>>> text
'Put several strings within parentheses to have them joined together.'
```

This only works with two literals though, not with variables or expressions:

```
>>>
>>> prefix = 'Py'
>>> prefix 'thon'  # can't concatenate a variable and a string literal
  File "<stdin>", line 1
    prefix 'thon'
           ^
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
  File "<stdin>", line 1
    ('un' * 3) 'ium'
               ^
SyntaxError: invalid syntax
```

If you want to concatenate variables or a variable and a literal, use +:

```
>>>
```

```
>>> prefix + 'thon'
'Python'
```

Strings can be indexed (subscripted), with the first character having index 0. There is no separate character type; a character is simply a string of size one:

```
>>>
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
```

Indices may also be negative numbers, to start counting from the right:

```
>>>
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'
```

Note that since -0 is the same as 0, negative indices start from -1.

In addition to indexing, slicing is also supported. While indexing is used to obtain individual characters, slicing allows you to obtain substring:

```
>>>
>>> word[0:2]  # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5 (excluded)
'tho'
```

Note how the start is always included, and the end always excluded. This makes sure that s[:i] + s[i:] is always equal to s:

```
>>>
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

Slice indices have useful defaults; an omitted first index defaults to zero, an omitted second index defaults to the size of the string being sliced.

```
>>>
>>> word[:2]   # character from the beginning to position 2 (excluded)
'Py'
>>> word[4:]   # characters from position 4 (included) to the end
'on'
>>> word[-2:]  # characters from the second-last (included) to the end
'on'
```

One way to remember how slices work is to think of the indices as pointing between characters, with the left edge of the first character numbered 0. Then the right edge of the last character of a string of n characters has index n, for example:

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

The first row of numbers gives the position of the indices 0...6 in the string; the second row gives the corresponding negative indices. The slice from i to j consists of all characters between the edges labeled i and j, respectively.

For non-negative indices, the length of a slice is the difference of the indices, if both are within bounds. For example, the length of word[1:3] is 2.

Attempting to use an index that is too large will result in an error:

```
>>>
```

```
>>> word[42]  # the word only has 6 characters
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

However, out of range slice indexes are handled gracefully when used for slicing:

```
>>>
>>> word[4:42]
'on'
>>> word[42:]
''
```

Python strings cannot be changed — they are immutable. Therefore, assigning to an indexed position in the string results in an error:

```
>>>
>>> word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

If you need a different string, you should create a new one:

```
>>>
>>> 'J' + word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```

The built-in function len() returns the length of a string:

>>>

>>> s = 'supercalifragilisticexpialidocious'

>>> len(s)

34""")

#LISTS

elif "Lists" in usertopop or "lists" in usertopop or "List" in usertopop or "list" in usertopop is True:

print("""

3.1.3. Lists

Python knows a number of compound data types, used to group together other values. The most versatile is the list, which can be written as a list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

>>>

>>> squares = [1, 4, 9, 16, 25]

>>> squares

[1, 4, 9, 16, 25]

Like strings (and all other built-in sequence types), lists can be indexed and sliced:

>>>

>>> squares[0] # indexing returns the item

1

>>> squares[-1]

25

>>> squares[-3:] # slicing returns a new list

[9, 16, 25]

All slice operations return a new list containing the requested elements. This means that the following slice returns a shallow copy of the list:

>>>

>>> squares[:]

[1, 4, 9, 16, 25]

Lists also support operations like concatenation:

>>>

>>> squares + [36, 49, 64, 81, 100]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Unlike strings, which are immutable, lists are a mutable type, i.e. it is possible to change their content:

>>>

>>> cubes = [1, 8, 27, 65, 125] # something's wrong here

>>> 4 ** 3 # the cube of 4 is 64, not 65!

64

>>> cubes[3] = 64 # replace the wrong value

>>> cubes

[1, 8, 27, 64, 125]

You can also add new items at the end of the list, by using the append() method (we will see more about methods later):

>>>

>>> cubes.append(216) # add the cube of 6

>>> cubes.append(7 ** 3) # and the cube of 7

>>> cubes

[1, 8, 27, 64, 125, 216, 343]

Assignment to slices is also possible, and this can even change the size of the list or clear it entirely:

>>>

>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

>>> letters

['a', 'b', 'c', 'd', 'e', 'f', 'g']

>>> # replace some values

```
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # clear the list by replacing all the elements with an empty list
>>> letters[:] = []
>>> letters
[]
```

The built-in function len() also applies to lists:

```
>>>
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

It is possible to nest lists (create lists containing other lists), for example:

```
>>>
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
""")
```

```
#LOOPING ie for and while
    elif "Looping" in usertopop or "looping" in usertopop is True:
        print("""
The "for" statement
*******************

The "for" statement is used to iterate over the elements of a sequence
(such as a string, tuple or list) or other iterable object:

  for_stmt ::= "for" target_list "in" expression_list ":" suite
               ["else" ":" suite]

The expression list is evaluated once; it should yield an iterable
object.  An iterator is created for the result of the
"expression_list".  The suite is then executed once for each item
provided by the iterator, in the order returned by the iterator.  Each
item in turn is assigned to the target list using the standard rules
for assignments (see Assignment statements), and then the suite is
executed.  When the items are exhausted (which is immediately when the
sequence is empty or an iterator raises a "StopIteration" exception),
the suite in the "else" clause, if present, is executed, and the loop
terminates.

A "break" statement executed in the first suite terminates the loop
without executing the "else" clause's suite.  A "continue" statement
executed in the first suite skips the rest of the suite and continues
with the next item, or with the "else" clause if there is no next
item.
```

The for-loop makes assignments to the variables in the target list. This overwrites all previous assignments to those variables including those made in the suite of the for-loop:

```
for i in range(10):
    print(i)
    i = 5           # this will not affect the for-loop
                    # because i will be overwritten with the next
                    # index in the range
```

Names in the target list are not deleted when the loop is finished, but if the sequence is empty, they will not have been assigned to at all by the loop. Hint: the built-in function "range()" returns an iterator of integers suitable to emulate the effect of Pascal's "for i := a to b do"; e.g., "list(range(3))" returns the list "[0, 1, 2]".

Note: There is a subtlety when the sequence is being modified by the loop (this can only occur for mutable sequences, e.g. lists). An internal counter is used to keep track of which item is used next, and this is incremented on each iteration. When this counter has reached the length of the sequence the loop terminates. This means that if the suite deletes the current (or a previous) item from the sequence, the next item will be skipped (since it gets the index of the current item which has already been treated). Likewise, if the suite inserts an item in the sequence before the current item, the current item will be treated again the next time through the loop. This can lead to nasty bugs that can be avoided by making a

temporary copy using a slice of the whole sequence, e.g.,

```
    for x in a[:]:
        if x < 0: a.remove(x)
```

The "while" statement
****

The "while" statement is used for repeated execution as long as an expression is true:

```
  while_stmt ::= "while" expression ":" suite
                 ["else" ":" suite]
```

This repeatedly tests the expression and, if it is true, executes the first suite; if the expression is false (which may be the first time it is tested) the suite of the "else" clause, if present, is executed and the loop terminates.

A "break" statement executed in the first suite terminates the loop without executing the "else" clause's suite. A "continue" statement executed in the first suite skips the rest of the suite and goes back to testing the expression.

Related help topics: break, continue, if, TRUTHVALUE
""")
#TUPLES
    elif "Tuples" in usertopop or "tuples" in usertopop is True:
        print("""

Tuples and Sequences

We saw that lists and strings have many common properties, such as indexing and slicing operations. They are two examples of sequence data types (see Sequence Types — list, tuple, range). Since Python is an evolving language, other sequence data types may be added. There is also another standard sequence data type: the tuple.

A tuple consists of a number of values separated by commas, for instance:

```
>>>
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

As you see, on output tuples are always enclosed in parentheses, so that nested tuples are interpreted correctly; they may be input with or without surrounding parentheses,

although often parentheses are necessary anyway (if the tuple is part of a larger expression). It is not possible to assign to the individual items of a tuple, however it is possible to create tuples which contain mutable objects, such as lists.

Though tuples may seem similar to lists, they are often used in different situations and for different purposes. Tuples are immutable, and usually contain a heterogeneous sequence of elements that are accessed via unpacking (see later in this section) or indexing (or even by attribute in the case of namedtuples). Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list.

A special problem is the construction of tuples containing 0 or 1 items: the syntax has some extra quirks to accommodate these. Empty tuples are constructed by an empty pair of parentheses; a tuple with one item is constructed by following a value with a comma (it is not sufficient to enclose a single value in parentheses). Ugly, but effective. For example:

```
>>>
>>> empty = ()
>>> singleton = 'hello',    # <-- note trailing comma
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)
```

The statement t = 12345, 54321, 'hello!' is an example of tuple packing: the values 12345, 54321 and 'hello!' are packed together in a tuple. The reverse operation is also possible:

```
>>>
>>> x, y, z = t
```

This is called, appropriately enough, sequence unpacking and works for any sequence on the right-hand side. Sequence unpacking requires that there are as many variables on the left side of the equals sign as there are elements in the sequence. Note that multiple assignment is really just a combination of tuple packing and sequence unpacking.""")

elif "Dictionaries" in usertopop or "dictionaries" in usertopop is True:

print("""

Dictionaries

Another useful data type built into Python is the dictionary (see Mapping Types — dict). 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. You can’t use lists 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. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.

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:

```
>>>
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

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

```
>>>
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

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

```
>>>
```

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

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

```
>>>
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'guido': 4127, 'jack': 4098}""")
   elif "If" in usertopop or "if" in usertopop is True:
      print("""
if Statements
Perhaps the most well-known statement type is the if statement. For example:

>>>
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

There can be zero or more elif parts, and the else part is optional. The keyword ‘elif’ is short for ‘else if’, and is useful to avoid excessive indentation. An if ... elif ... elif ... sequence is a substitute for the switch or case statements found in other languages.""")

```
#GOOGLE SEARCH CODE
  elif "Search" in userop or "search" in userop or "google" in userop or "Google" in userop or "Link" in userop or "link" in userop is True:
      s=input("Enter a Topic name")
      print("http://www.google.com/search?q=",s,"&oq=",s)
# TIC TAC TOE CODE
  elif "Tic tac toe" in userop or "tic tac toe" in userop or "tic" in  userop or "tac" in userop or "toe" in userop is True:
      import random
      import sys
      board=[i for i in range(0,9)]
      player, computer = '','
      moves=((1,7,3,9),(5,),(2,4,6,8))
      winners=((0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8),(0,4,8),(2,4,6))
      tab=range(1,10)
      def print_board():
         x=1
         for i in board:
            end = ' | '
            if x%3 == 0:
               end = ' \n'
               if i != 1: end+='---------\n';
            char=' '
            if i in ('X','O'): char=i;
            x+=1
            print(char,end=end)
```

```
def select_char():
    chars=('X','O')
    if random.randint(0,1) == 0:
        return chars[::-1]
    return chars
def can_move(brd, player, move):
    if move in tab and brd[move-1] == move-1:
        return True
    return False
def can_win(brd, player, move):
    places=[]
    x=0
    for i in brd:
        if i == player: places.append(x);
        x+=1
    win=True
    for tup in winners:
        win=True
        for ix in tup:
            if brd[ix] != player:
                win=False
                break
        if win == True:
            break
    return win
def make_move(brd, player, move, undo=False):
    if can_move(brd, player, move):
        brd[move-1] = player
```

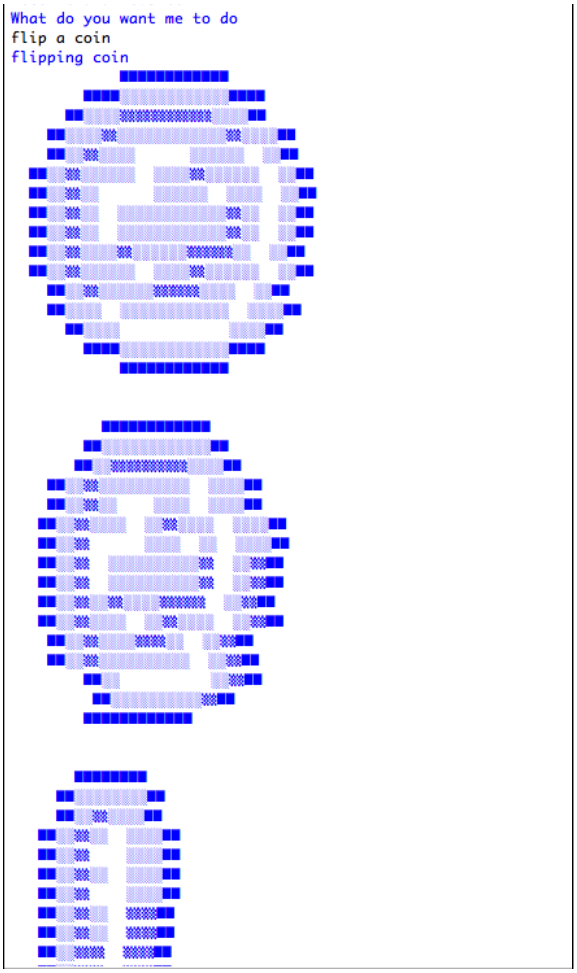
```
            win=can_win(brd, player, move)
            if undo:
                brd[move-1] = move-1
            return (True, win)
        return (False, False)
    def computer_move():
        move=-1
        for i in range(1,10):
            if make_move(board, computer, i, True)[1]:
                move=i
                break
        if move == -1:
            for i in range(1,10):
                if make_move(board, player, i, True)[1]:
                    move=i
                    break
        if move == -1:
            for tup in moves:
                for mv in tup:
                    if move == -1 and can_move(board, computer, mv):
                        move=mv
                        break
        return make_move(board, computer, move)
    def space_exist():
        return board.count('X') + board.count('O') != 9
    player, computer = select_char()
    print('Player is [%s] and computer is [%s]' % (player, computer))
    result='%%% Deuce ! %%%'
```

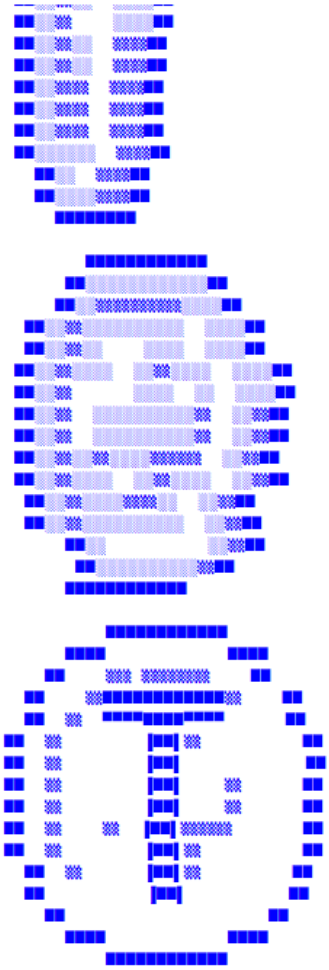
```
        while space_exist():
            print_board()
            print('# Make your move ! [1-9] : ', end='')
            move = int(input())
            moved, won = make_move(board, player, move)
            if not moved:
                print(' >> Invalid number ! Try again !')
                continue
            if won:
                result='*** Congratulations ! You won ! ***'
                break
            elif computer_move()[1]:
                result='=== You lose ! =='
                break;
        print_board()
        print(result)
# CALENDAR CODE
    elif "Calendar" in userop or "calendar" in userop or "month" in  userop or "Month" in userop or "year" in userop is True:
        import calendar
        y = int(input("Input the year : "))
        m = int(input("Input the month : "))
        print(calendar.month(y, m))
# TIME CODE
    elif "Time" in userop or "time" in userop is True:
        from datetime import datetime
        now = datetime.now()
        print("now =", now)
```

```
        dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
        print("date and time =", dt_string)
# WEBSITE RECOMMENDER
    elif "website" in userop or "Website" in userop or "Recommend" in userop or "recommend" in userop is True:
        web=["Waitbutwhy.com","Theuselessweb.com","""https://asoftmurmur.com/ Just play around
with the sliders to achieve your perfectly calming background noise. It's
impossible not to try every one and each has its own unique charm""","""https://stars.chromeexperiments.com/Love galaxies and stars? 100,00 Stars is the website which can take you through them. It also offers a tour from Son to other edges and offers information to learn about them.""","http://www.vsauce.com/#/","""https://dinosaurpictures.org/ancient-earth#170
If you want to find out how the earth looked a couple of million years ago, this is the website you need to check out. It uses historical data based on geographic analysis to recreate. Once it generates, you can turn the globe around to see how the continents were set up during that time.""","""https://www.thispersondoesnotexist.com/
AI or Artificial Intelligence have come so far that it can create fake people that don't exist. Every time you refresh the website, you get to see a new non-existent person."""]
        webs=random.randint(0,4)
        print_slow(web[webs])
        print("")
```

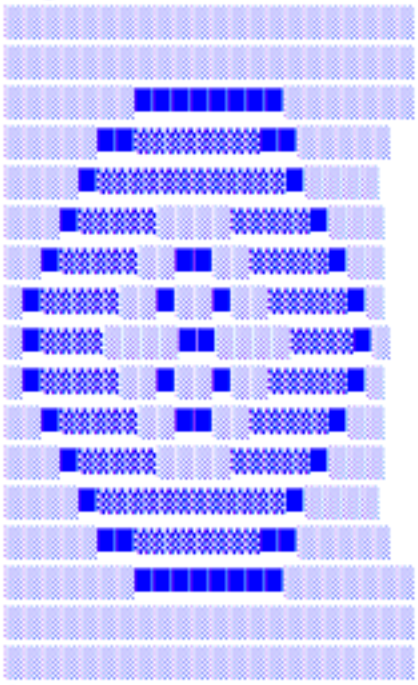
Screenshots

```
Hi! :)
Im Surge Python assistant
I can do things like
Flip a coin
Help you with Python code
Play Tic tac toe
Recommend a website
What do you want me to do
```


What do you want me to do
flip a coin
flipping coin



```
What do you want me to do
Magic 8 ball
This is the magic 8 ball ask a question or type quit to quit
Enter Question here : Will it rain tomorrow?
Shaking the ball
```



```
What do you want me to do
help in python code
Surge can help you with py topics like...
Numbers,String,Lists,Looping,Tuples,Dictionaries,if statements
What would you like to learn today?
```

```
What do you want me to do
tic tac toe
Player is [O] and computer is [X]
  |   |
---------
  |   |
---------
  |   |
---------
# Make your move ! [1-9] : 5
X |   |
---------
  | O |
---------
  |   |
---------
# Make your move ! [1-9] : 9
X |   |
---------
  | O |
---------
X |   | O
---------
# Make your move ! [1-9] : 4
X |   |
---------
O | O | X
---------
X |   | O
---------
# Make your move ! [1-9] : 2
X | O |
---------
O | O | X
---------
X | X | O
---------
# Make your move ! [1-9] : 3
X | O | O
---------
O | O | X
---------
X | X | O
---------
%%% Deuce ! %%%
```

```
What do you want me to do
calendar
Input the year : 2020
Input the month : 1
    January 2020
Mo Tu We Th Fr Sa Su
        1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

```
What do you want me to do
Recommend a website
https://stars.chromeexperiments.com/Love galaxies and stars? 100,00 Stars is the website which can take you through them. It also offers a tour from Son to other edges and offers information to lear n about them.
```