## Python Keywords

## Keywords are the reserved words in Python.

We cannot use a keyword as a [variable name](https://www.programiz.com/python-programming/variables-datatypes), [function](https://www.programiz.com/python-programming/function) name or any other identifier. They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive.

There are 33 keywords in Python 3.7. This number can vary slightly in the course of time.

All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords is given below.

Keywords in Python

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False | class | finally | is | return |
| None | continue | for | lambda | try |
| True | def | from | nonlocal | while |
| and | del | global | not | with |
| as | elif | if | or | yield |
| assert | else | import | pass |  |
| break | except | in | raise |  |

Looking at all the keywords at once and trying to figure out what they mean might be overwhelming.

If you want to have an overview, here is the complete [list of all the keywords](https://www.programiz.com/python-programming/keyword-list) with examples.

## Python Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

### Rules for writing identifiers

1. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore \_.Names like myClass, var\_1 and print\_this\_to\_screen, all are valid example.
2. An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
3. Keywords cannot be used as identifiers.

>>> global = 1

File "<interactive input>", line 1

global = 1

^

SyntaxError: invalid syntax

## 4.We cannot use special symbols like !, @, #, $, % etc. in our identifier.

>>> a@ = 0

File "<interactive input>", line 1

a@ = 0

^

SyntaxError: invalid syntax

## 5.Identifier can be of any length.

### **Things to Remember**

Python is a case-sensitive language. This means, Variable and variable are not the same. Always name identifiers that make sense.

While, c = 10 is valid. Writing count = 10 would make more sense and it would be easier to figure out what it does even when you look at your code after a long gap.

Multiple words can be separated using an underscore, this\_is\_a\_long\_variable.

## **Python Statement**

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement. if statement, for statement, while statement etc. are other kinds of statements which will be discussed later.

### **Multi-line statement**

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

a = 1 + 2 + 3 + \

4 + 5 + 6 + \

7 + 8 + 9

This is explicit line continuation. In Python, line continuation is implied inside parentheses ( ), brackets [ ] and braces { }. For instance, we can implement the above multi-line statement as

a = (1 + 2 + 3 +

4 + 5 + 6 +

7 + 8 + 9)

colors = ['red',

'blue',

'green']

We could also put multiple statements in a single line using semicolons, as follows

a = 1; b = 2; c = 3

## **Python Indentation**

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

A code block (body of a [function](https://www.programiz.com/python-programming/function), [loop](https://www.programiz.com/python-programming/for-loop) etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

Generally four whitespaces are used for indentation and is preferred over tabs. Here is an example.

## for i in range(1,11):

## print(i)

## if i == 5:

## break

The enforcement of indentation in Python makes the code look neat and clean. This results into Python programs that look similar and consistent.

Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable. For example:

if True:

print('Hello')

a = 5

and

if True: print('Hello'); a = 5

both are valid and do the same thing. But the former style is clearer.

Incorrect indentation will result into IndentationError

## Python Comments

Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out. You might forget the key details of the program you just wrote in a month's time. So taking time to explain these concepts in form of comments is always fruitful.

In Python, we use the hash (#) symbol to start writing a comment.

It extends up to the newline character. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

#This is a comment

#print out Hello

print('Hello')

### Multi-line comments

If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line. For example:

#This is a long comment

#and it extends

#to multiple lines

Another way of doing this is to use triple quotes, either ''' or """.

These triple quotes are generally used for multi-line strings. But they can be used as multi-line comment as well. Unless they are not docstrings, they do not generate any extra code.

"""This is also a

perfect example of

multi-line comments"""

### **Docstring in Python**

Docstring is short for documentation string.

It is a [string](https://www.programiz.com/python-programming/string) that occurs as the first statement in a module, function, class, or method definition. We must write what a function/class does in the docstring.

Triple quotes are used while writing docstrings. For example:

def double(num):

"""Function to double the value"""

return 2\*num

Docstring is available to us as the attribute \_\_doc\_\_ of the function. Issue the following code in shell once you run the above program.

>>> print(double.\_\_doc\_\_)

Function to double the value

### **Literal Collections**

There are four different literal collections List literals, Tuple literals, Dict literals, and Set literals.

#### Example 10: How to use literals collections in Python?

fruits = ["apple", "mango", "orange"] #list

numbers = (1, 2, 3) #tuple

alphabets = {'a':'apple', 'b':'ball', 'c':'cat'} #dictionary

vowels = {'a', 'e', 'i' , 'o', 'u'} #set

print(fruits)

print(numbers)

print(alphabets)

print(vowels)

**When you run the program, the output will be:**

['apple', 'mango', 'orange']

(1, 2, 3)

{'a': 'apple', 'b': 'ball', 'c': 'cat'}

{'e', 'a', 'o', 'i', 'u'}

## Data types in Python

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

### Python Numbers

Integers, floating point numbers and complex numbers falls under [Python numbers](https://www.programiz.com/python-programming/numbers) category. They are defined as int, float and complex class in Python.

We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

Example:

a = 5

print(a, "is of type", type(a))

a = 2.0

print(a, "is of type", type(a))

a = 1+2j

print(a, "is complex number?", isinstance(1+2j,complex))

output:

5 is of type <class 'int'>

2.0 is of type <class 'float'>

(1+2j) is complex number? True

Integers can be of any length, it is only limited by the memory available.

A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number.

Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. Here are some examples.

>>> a = 1234567890123456789

>>> a

1234567890123456789

>>> b = 0.1234567890123456789

>>> b

0.12345678901234568

>>> c = 1+2j

>>> c

(1+2j)

### Python List

[List](https://www.programiz.com/python-programming/list) is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].

>>> a = [1, 2.2, 'python']

We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

**Example**:

a = [5,10,15,20,25,30,35,40]

# a[2] = 15

print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]

print("a[0:3] = ", a[0:3])

# a[5:] = [30, 35, 40]

print("a[5:] = ", a[5:])

**output**:

a[2] = 15

a[0:3] = [5, 10, 15]

a[5:] = [30, 35, 40]

Lists are mutable, meaning, value of elements of a list can be altered.

>>> a = [1,2,3]

>>> a[2]=4

>>> a

[1, 2, 4]

### Python Tuple

[Tuple](https://www.programiz.com/python-programming/tuple) is an ordered sequence of items same as list.The only difference is that tuples are immutable. Tuples once created cannot be modified.

Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically.

It is defined within parentheses () where items are separated by commas.

>>> t = (5,'program', 1+3j)

example:

t = (5,'program', 1+3j)

# t[1] = 'program'

print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))

print("t[0:3] = ", t[0:3])

# Generates error

# Tuples are immutable

t[0] = 10

### Python Strings

[String](https://www.programiz.com/python-programming/string) is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

>>> s = "This is a string"

>>> s = '''a multiline

Like list and tuple, slicing operator [ ] can be used with string. Strings are immutable.

**Example :**

s = 'Hello world!'

# s[4] = 'o'

print("s[4] = ", s[4])

# s[6:11] = 'world'

print("s[6:11] = ", s[6:11])

# Generates error

# Strings are immutable in Python

s[5] ='d'

**output** :

s[4] = o

s[6:11] = world

Traceback (most recent call last):

File "<stdin>", line 11, in <module>

s[5] ='d'

TypeError: 'str' object does not support item assignment

### Python Set

[Set](https://www.programiz.com/python-programming/set) is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

Example :

a = {5,2,3,1,4}

# printing set variable

print("a = ", a)

# data type of variable a

print(type(a))

Output:

a = {1, 2, 3, 4, 5}

<class 'set'>

We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates.

>>> a = {1,2,2,3,3,3}

>>> a

{1, 2, 3}

Since, set are unordered collection, indexing has no meaning. Hence the slicing operator [] does not work.

>>> a = {1,2,3}

>>> a[1]

Traceback (most recent call last):

File "<string>", line 301, in runcode

File "<interactive input>", line 1, in <module>

TypeError: 'set' object does not support indexing

### Python Dictionary

[Dictionary](https://www.programiz.com/python-programming/dictionary) is an unordered collection of key-value pairs.

It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.

In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

>>> d = {1:'value','key':2}

>>> type(d)

<class 'dict'>

We use key to retrieve the respective value. But not the other way around.

**Example:**

d = {1:'value','key':2}

print(type(d))

print("d[1] = ", d[1]);

print("d['key'] = ", d['key']);

# Generates error

print("d[2] = ", d[2]);

Output:

<class 'dict'>

d[1] = value

d['key'] = 2

Traceback (most recent call last):

File "<stdin>", line 9, in <module>

print("d[2] = ", d[2]);

KeyError: 2

## Type Conversion:

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

1. Implicit Type Conversion
2. Explicit Type Conversion

## Implicit Type Conversion:

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

Let's see an example where Python promotes conversion of lower datatype (integer) to higher data type (float) to avoid data loss.

num\_int = 123

num\_flo = 1.23

num\_new = num\_int + num\_flo

print("datatype of num\_int:",type(num\_int))

print("datatype of num\_flo:",type(num\_flo))

print("Value of num\_new:",num\_new)

print("datatype of num\_new:",type(num\_new))

**Output**:

datatype of num\_int: <class 'int'>

datatype of num\_flo: <class 'float'>

Value of num\_new: 124.23

datatype of num\_new: <class 'float'>

In the above program,

* We add two variables num\_int and num\_flo, storing the value in num\_new.
* We will look at the data type of all three objects respectively.
* In the output we can see the datatype of num\_int is an integer, datatype of num\_flo is a float.
* Also, we can see the num\_new has float data type because Python always converts smaller data type to larger data type to avoid the loss of data.

Now, let's try adding a string and an integer, and see how Python treats it.

### Example 2: Addition of string(higher) data type and integer(lower) datatype

**num\_int = 123**

**num\_str = "456"**

**print("Data type of num\_int:",type(num\_int))**

**print("Data type of num\_str:",type(num\_str))**

**print(num\_int+num\_str)**

Output:

**Data type of num\_int: <class 'int'>**

**Data type of num\_str: <class 'str'>**

**Traceback (most recent call last):**

**File "<stdin>", line 7, in <module>**

**print(num\_int+num\_str)**

**TypeError: unsupported operand type(s) for +: 'int' and 'str'**

**In the above program,**

* We add two variable num\_int and num\_str.
* As we can see from the output, we got typeerror. Python is not able use Implicit Conversion in such condition.
* However Python has the solution for this type of situation which is know as Explicit Conversion.

## **Explicit Type Conversion:**

In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.

This type conversion is also called typecasting because the user casts (change) the data type of the objects

**Syntax :**

**(required\_datatype)(expression)**

Typecasting can be done by assigning the required data type function to the expression.

num\_int = 123

num\_str = "456"

print("Data type of num\_int:",type(num\_int))

print("Data type of num\_str before Type Casting:",type(num\_str))

num\_str = int(num\_str)

print("Data type of num\_str after Type Casting:",type(num\_str))

num\_sum = num\_int + num\_str

print("Sum of num\_int and num\_str:",num\_sum)

print("Data type of the sum:",type(num\_sum))

**output:**

**Data type of num\_int: <class 'int'>**

**Data type of num\_str before Type Casting: <class 'str'>**

**Data type of num\_str after Type Casting: <class 'int'>**

**Sum of num\_int and num\_str: 579**

**Data type of the sum: <class 'int'>**

**In above program,**

* We add num\_str and num\_int variable.
* We converted num\_str from string(higher) to integer(lower) type using int() function to perform the addition.
* After converting num\_str to a integer value Python is able to add these two variable.
* We got the num\_sum value and data type to be integer.

## **Key Points to Remember:**

1. Type Conversion is the conversion of object from one data type to another data type.
2. Implicit Type Conversion is automatically performed by the Python interpreter.
3. Python avoids the loss of data in Implicit Type Conversion.
4. Explicit Type Conversion is also called Type Casting, the data types of object are converted using predefined function by user.
5. In Type Casting loss of data may occur as we enforce the object to specific data type.

# Python Functions

In Python , function is a group of related statements that perform a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes code reusable.

### Syntax of Function

def function\_name(parameters):

"""docstring"""

statement(s

Above shown is a function definition which consists of following components.

1. Keyword def marks the start of function header.
2. A function name to uniquely identify it. Function naming follows the same [rules of writing identifiers in Python](https://www.programiz.com/python-programming/keywords-identifier" \l "rules).
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
7. An optional return statement to return a value from the function.

Example:

def greet(name):

"""This function greets to

the person passed in as

parameter"""

print("Hello, " + name + ". Good morning!")

greet("AKBAR")

output:

Hello, AKBAR. Good morning!

### How to call a function in python?

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

## Docstring

The first string after the function header is called the docstring and is short for documentation string. It is used to explain in brief, what a function does.

Although optional, documentation is a good programming practice. Unless you can remember what you had for dinner last week, always document your code.

In the above example, we have a docstring immediately below the function header. We generally use triple quotes so that docstring can extend up to multiple lines. This string is available to us as \_\_doc\_\_ attribute of the function.

For example:

Try running the following into the Python shell to see the output.

>>> print(greet.\_\_doc\_\_)

This function greets to

the person passed into the

name parameter

example:

def greet(name):

"""This function greets to

the person passed in as

parameter"""

print("Hello, " + name + ". Good morning!")

greet("AKBAR")

print(greet.\_\_doc\_\_)

output:

Hello, AKBAR. Good morning!

This function greets to

the person passed in as

parameter

## The return statement

The return statement is used to exit a function and go back to the place from where it was called.

### Syntax of return

return [expression\_list]

example:

def absolute\_value(num):

"""This function returns the absolute

value of the entered number"""

if num >= 0:

return num

else:

return -num

# Output: 2

print(absolute\_value(2))

# Output: 4

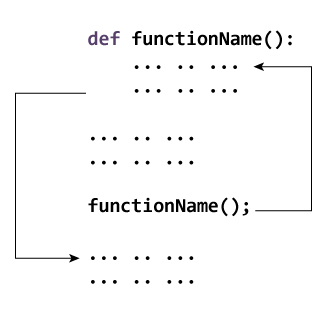
print(absolute\_value(-4))

output:

2

4

## How Function works in Python?



## Scope and Lifetime of variables

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.

Lifetime of a variable is the period throughout which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes.

They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Here is an example to illustrate the scope of a variable inside a function.

Example:

def my\_func():

x = 10

print("Value inside function:",x)

x = 20

my\_func()

print("Value outside function:",x)

output:

Value inside function: 10

Value outside function: 20

Here, we can see that the value of x is 20 initially. Even though the function my\_func() changed the value of x to 10, it did not effect the value outside the function.

This is because the variable x inside the function is different (local to the function) from the one outside. Although they have same names, they are two different variables with different scope.

On the other hand, variables outside of the function are visible from inside. They have a global scope.

We can read these values from inside the function but cannot change (write) them. In order to modify the value of variables outside the function, they must be declared as global variables using the keyword global.

## Types of Functions

Basically, we can divide functions into the following two types:

1. [Built-in functions](https://www.programiz.com/python-programming/built-in-function) - Functions that are built into Python.
2. [User-defined functions](https://www.programiz.com/python-programming/user-defined-function) - Functions defined by the users themselves.

# Python Function Arguments

In Python, you can define a function that takes variable number of arguments. You will learn to define such functions using default, keyword and arbitrary arguments in this article.

## Arguments

In [user-defined function](https://www.programiz.com/python-programming/user-defined-function) topic, we learned about defining a function and calling it. Otherwise, the function call will result into an error. Here is an example.

Ex:def greet(name,msg):

"""This function greets to

the person with the provided message"""

print("Hello",name + ', ' + msg)

greet("Monica","Good morning!")

print(greet.\_\_doc\_\_)

output:

Hello Monica, Good morning!

This function greets to

the person with the provided message

### Python Default Arguments

Function arguments can have default values in Python.

We can provide a default value to an argument by using the assignment operator (=). Here is an example

def greet(name, msg = "Good morning!"):

"""

This function greets to

the person with the

provided message.

If message is not provided,

it defaults to "Good

morning!"

"""

print("Hello",name + ', ' + msg)

greet("Kate")

greet("Bruce","How do you do?")

Output:

Hello Kate, Good morning!

Hello Bruce, How do you do?

In this function, the parameter name does not have a default value and is required (mandatory) during a call.

On the other hand, the parameter msg has a default value of "Good morning!". So, it is optional during a call. If a value is provided, it will overwrite the default value.

Any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

This means to say, non-default arguments cannot follow default arguments. For example, if we had defined the function header above as:

### Python Keyword Arguments

When we call a function with some values, these values get assigned to the arguments according to their position.

For example, in the above function greet(), when we called it as greet("Bruce","How do you do?"), the value "Bruce" gets assigned to the argument name and similarly "How do you do?" to msg.

Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed. Following calls to the above function are all valid and produce the same result.

### Python Arbitrary Arguments

Sometimes, we do not know in advance the number of arguments that will be passed into a function.Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.

In the function definition we use an asterisk (\*) before the parameter name to denote this kind of argument. Here is an example.

def greet(\*names):

"""This function greets all

the person in the names tuple."""

# names is a tuple with arguments

for name in names:

print("Hello",name)

greet("Monica","Luke","Steve","John")

Output

Hello Monica

Hello Luke

Hello Steve

Hello John

Here, we have called the function with multiple arguments. These arguments get wrapped up into a tuple before being passed into the function. Inside the function, we use a for loop to retrieve all the arguments back.

## What is recursion in Python?

Recursion is the process of defining something in terms of itself.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

## Python Recursive Function

We know that in Python, a [function](https://www.programiz.com/python-programming/function) can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.

Following is an example of recursive function to find the factorial of an integer.

Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1\*2\*3\*4\*5\*6 = 720

# An example of a recursive function to

# find the factorial of a number

def calc\_factorial(x):

"""This is a recursive function

to find the factorial of an integer"""

if x == 1:

return 1

else:

return (x \* calc\_factorial(x-1))

num = 4

print("The factorial of", num, "is", calc\_factorial(num))

output:

The factorial of 4 is 24

In the above example, calc\_factorial() is a recursive functions as it calls itself.

When we call this function with a positive integer, it will recursively call itself by decreasing the number.

## Advantages of Recursion

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

## Disadvantages of Recursion

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

# **Python Anonymous/Lambda Function**

## What are lambda functions in Python?

In Python, anonymous function is a [function](https://www.programiz.com/python-programming/function) that is defined without a name.

While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.

Hence, anonymous functions are also called lambda functions

## How to use lambda Functions in Python?

A lambda function in python has the following syntax.

### Syntax of Lambda Function in python

lambda arguments: expression

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

# Program to show the use of lambda functions

double = lambda x: x \* 2

# Output: 10

print(double(5))

In the above program, lambda x: x \* 2 is the lambda function. Here x is the argument and x \* 2 is the expression that gets evaluated and returned.

This function has no name. It returns a function object which is assigned to the identifier double. We can now call it as a normal function. The statement

double = lambda x: x \* 2

is nearly the same as

def double(x):

return x \* 2

### Use of Lambda Function in python

We use lambda functions when we require a nameless function for a short period of time.

In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as [arguments](https://www.programiz.com/python-programming/function-argument)). Lambda functions are used along with built-in functions like filter(), map() etc.

#### Example use with filter()

The filter() function in Python takes in a function and a list as arguments.

The function is called with all the items in the list and a new list is returned which contains items for which the function evaluats to True.

Here is an example use of filter() function to filter out only even numbers from a list.

# Program to filter out only the even items from a list

my\_list = [1, 5, 4, 6, 8, 11, 3, 12]

new\_list = list(filter(lambda x: (x%2 == 0) , my\_list))

# Output: [4, 6, 8, 12]

print(new\_list)

#### Example use with map()

The map() function in Python takes in a function and a list.

The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

Here is an example use of map() function to double all the items in a list.

# Program to double each item in a list using map()

my\_list = [1, 5, 4, 6, 8, 11, 3, 12]

new\_list = list(map(lambda x: x \* 2 , my\_list))

# Output: [2, 10, 8, 12, 16, 22, 6, 24]

print(new\_list)

Output:

[2, 10, 8, 12, 16, 22, 6, 24]