PYTHON VARIABLE AND OBJECT TYPES

Partly from www.py4e.com

Constants

- Fixed values such as numbers, letters, and strings, are called “constants” because their value does not change

Numeric constants are as you expect

String constants use single quotes (') or double quotes (")

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

Reserved Words

You cannot use reserved words as variable names / identifiers

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

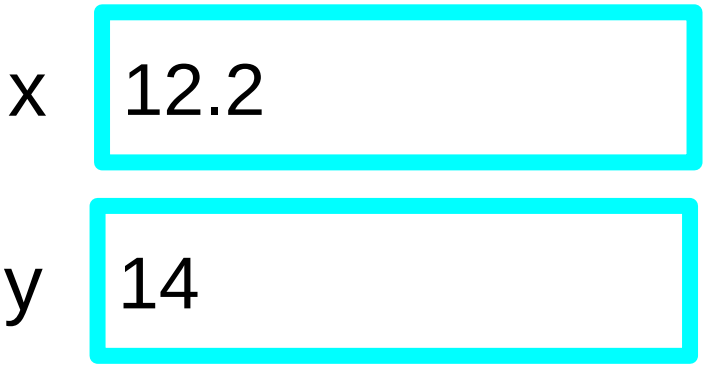
Variables

A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable ‘‘name’’

Programmers get to choose the names of the variables

You can change the contents of a variable in a later statement

```
x = 12.2
y = 14
```



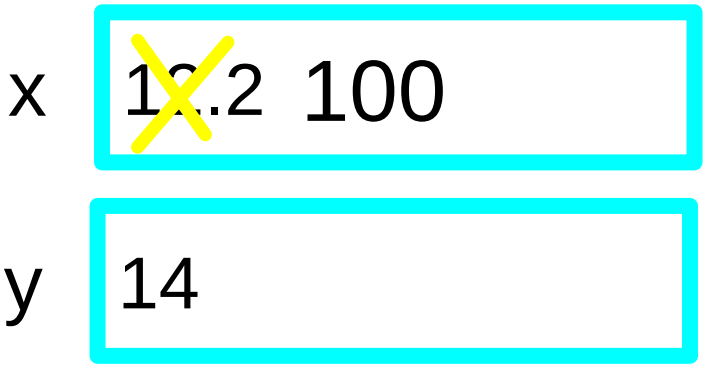
Variables

A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable ‘‘name’’

Programmers get to choose the names of the variables

You can change the contents of a variable in a later statement

```
x = 12.2
y = 14
x = 100
```



Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive

```
Good:     spam    eggs    spam23     _speed
Bad:      23spam      #sign   var.12
Different:     spam    Spam    SPAM
```

Mnemonic Variable Names

Since we programmers are given a choice in how we choose our variable names, there is a bit of “best practice”

We name variables to help us remember what we intend to store in them (“mnemonic” = “memory aid”)

This can confuse beginning students because well-named variables often “sound” so good that they must be keywords

http://en.wikipedia.org/wiki/Mnemonic

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

What is this bit of code doing?

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

What are these bits of code doing?

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print(x1q3p9afd)
```

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

What are these bits of code doing?

```
hours = 35.0
rate = 12.50
pay = hours * rate
print(pay)
```

Sentences or Lines

```
x = 2
x = x + 2
print(x)
```

← Assignment statement

← Assignment with expression

← Print statement

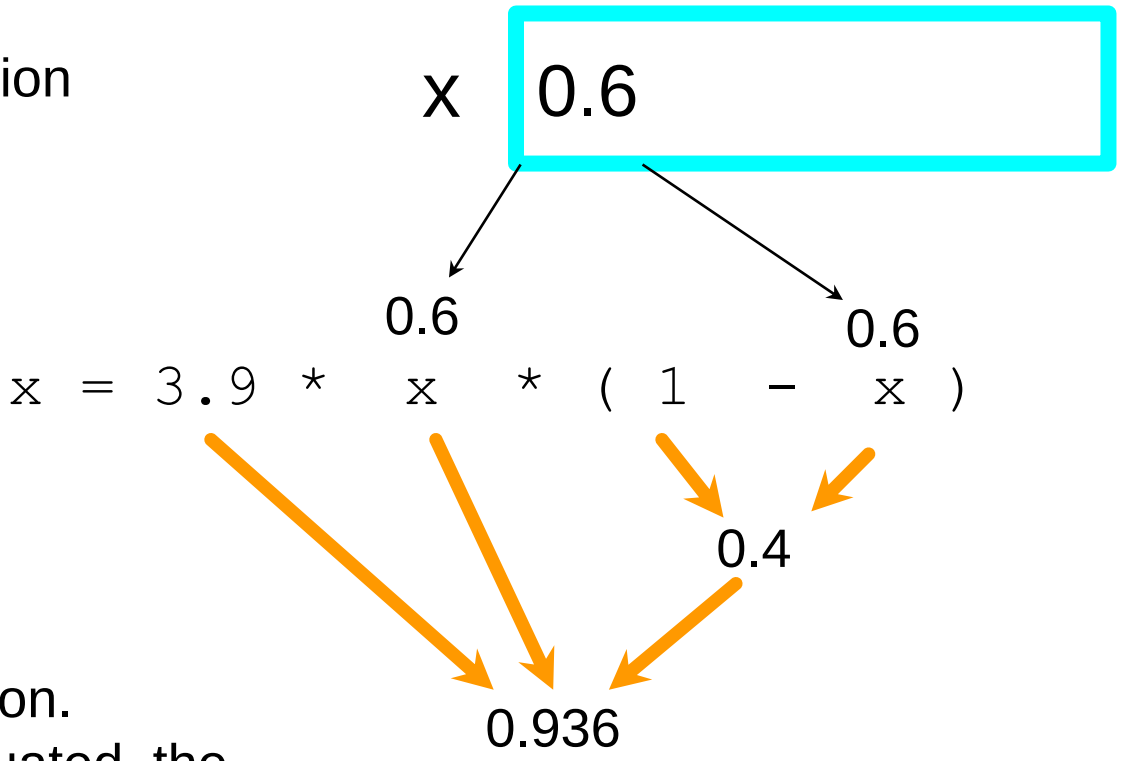
Variable Operator Constant Function

Assignment Statements

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right-hand side and a variable to store the result

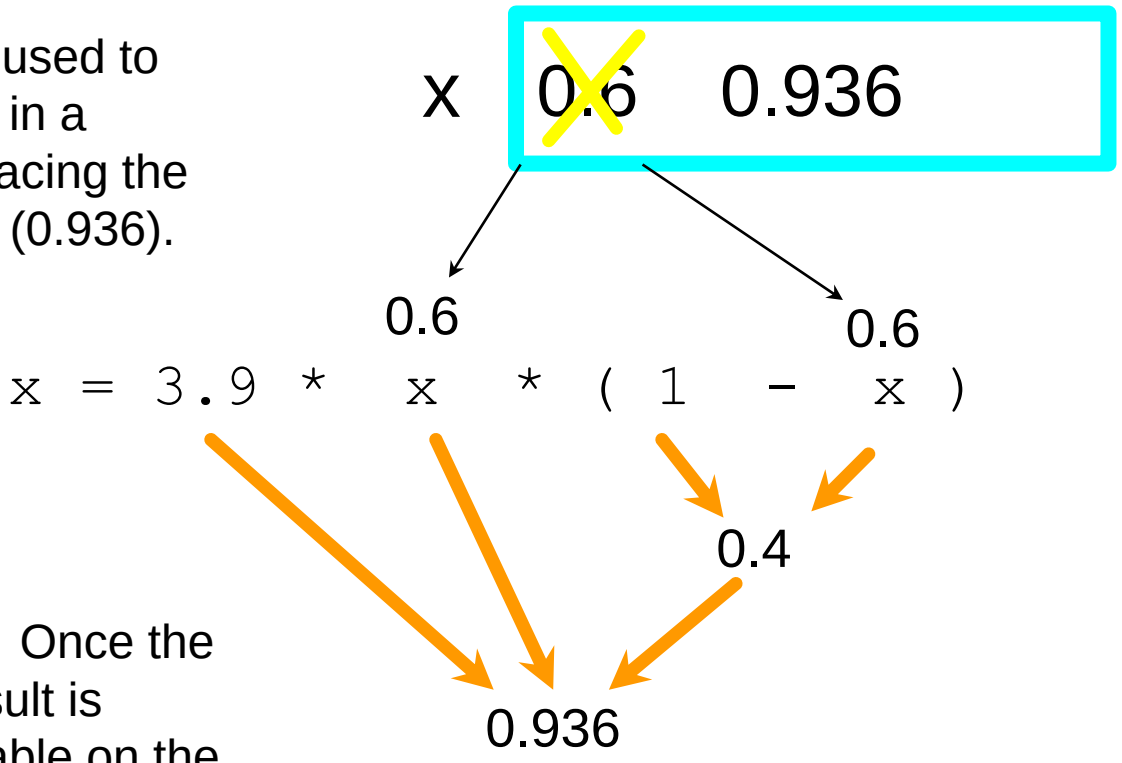
```
x = 3.9 * x * ( 1 - x )
```

A variable is a memory location used to store a value (0.6)



The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) x.

A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.936).



The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e., x).

Numeric Expressions

Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express the classic math operations

Asterisk is multiplication

Exponentiation (raise to a power) looks different than in math

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Numeric Expressions

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
```

```
    4 R 3
  ------
5 | 23
    20
  ----
     3
```

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Order of Evaluation

When we string operators together - Python must know which one to do first

This is called “operator precedence”

Which operator “takes precedence” over the others?

```
x = 1 + 2 * 3 - 4 / 5 ** 6
```

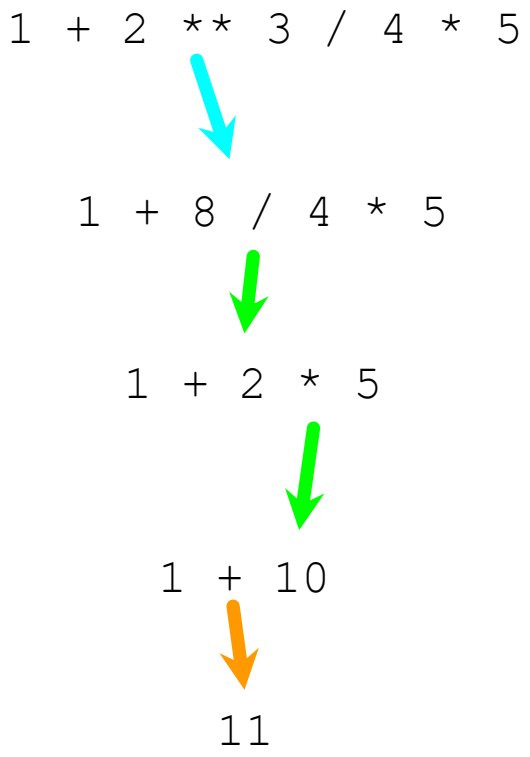
Operator Precedence Rules

Highest precedence rule to lowest precedence rule:

- Parentheses are always respected
- Exponentiation (raise to a power)
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Parenthesis
Power
Multiplication
Addition
Left to Right



Operator Precedence

Parenthesis
Power
Multiplication
Addition
Left to Right

Remember the rules top to bottom

When writing code - use parentheses

When writing code - keep mathematical expressions simple enough that they are easy to understand

Break long series of mathematical operations up to make them more clear

What Does “Type” Mean?

In Python variables, literals, and constants have a “type”

Python knows the difference between an integer number and a string

For example “+” means “addition” if something is a number and “concatenate” if something is a string

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hello ' + 'there'
>>> print(eee)
hello there
```

concatenate = put together

Type Matters

Python knows what “type” everything is

Some operations are prohibited

- You cannot “add 1” to a string

We can ask Python what type something is by using the type() function

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
File "<stdin>", line 1, in
<module>TypeError: Can't convert
'int' object to str implicitly
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
>>>
```

Several Types of Numbers

Numbers have two main types

- Integers are whole numbers:
  -14, -2, 0, 1, 100, 401233

- Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0

There are other number types - they are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class'float'>
>>>
```

Type Conversions

When you put an integer and floating point in an expression, the integer is implicitly converted to a float

You can control this with the built-in functions int() and float()

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class'float'>
>>>
```

Integer Division

Integer division produces a floating point result

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```

This was different in Python 2.x

String Conversions

You can also use int() and float() to convert between strings and integers

You will get an error if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object
to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: 'x'
```

User Input

We can instruct Python to pause and read data from the user using the input() function

The input() function returns a string

```
nam = input('Who are you? ')
print('Welcome', nam)
```

Who are you? Bob
Welcome Bob

Converting User Input

If we want to read a number from the user, we must convert it from a string to a number using a type conversion function

Later we will deal with bad input data

```
inp = input('Europe floor?')
usf = int(inp) + 1
print('Canada floor', usf)
```

Europe floor? 0
Canada floor 1

Comments in Python

Anything after a # is ignored by Python

Why comment?

- Describe what is going to happen in a sequence of code
- Document who wrote the code or other ancillary information
- Turn off a line of code - perhaps temporarily

```
# Get the name of the file and open it
name = input('Enter file:')
handle = open(name, 'r')

# Count word frequency
counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

# Find the most common word
bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

# All done
print(bigword, bigcount)
```

Exercise

Write a program to prompt the user for hours and rate per hour to compute gross pay.

```
Enter Hours: 35
Enter Rate: 2.75

Pay: 96.25
```

Introducing Python Object Types

- In Python, everything is an object
  - Built in
  - User created in Python
  - Externally created (say in C extension libraries)

- Python program structure:
  1. Programs are composed of modules.
  2. Modules contain statements.
  3. Statements contain expressions.
  4. Expressions create and process objects.

Core Objects

- Numbers → 1234, 3.1415, 3+4j, int, float, complex, bin, hex, oct.
- Strings → 'spam', "guido's", b'a\x01c'
- Lists → [1, [2, 'three'], 4]
- Dictionaries → {'food': 'spam', 'taste': 'yum'}
- Tuples → (1, 'spam', 4, 'U')
- Files → myfile = open('eggs', 'r')
- Sets → set('abc'), {'a', 'b', 'c'}
- Other core types → Booleans, None
- Remember that everything is an object!

Dynamic Type

- Python is **dynamically typed**
  - it keeps track of types for you automatically instead of requiring declaration code

- Python is **strongly typed**
  - Once you create an object, you bind its operation set for all time. i.e. you can perform only string operations on a string

Numbers

- Python’s core objects set includes: integers, floating-point, complex numbers, fixed-precision decimals, rational fractions
- Numbers support the normal mathematical operations. (+,-,*, **, etc.)

```
>>> 123 + 222                          # Integer addition
345
>>> 1.5 * 4                            # Floating-point multiplication
6.0
>>> 2 ** 100                           # 2 to the power 100
1267650600228229401496703205376
```

```
>>> len(str(2 ** 1000000))            # How many digits in a really BIG number?
301030
```

```
>>> 3.1415 * 2                         # repr: as code
6.2830000000000004
>>> print(3.1415 * 2)                  # str: user-friendly
6.283
```

Numeric modules

```
>>> import math
>>> math.pi
3.1415926535897931
>>> math.sqrt(85)
9.2195444572928871

>>> import random
>>> random.random()
0.59268735266273953
>>> random.choice([1, 2, 3, 4])
1
```

Q: How would you see all the methods in math?

Strings

- Strings are used for text as well as arbitrary collections of bytes.
- Strings are one form of Python "*sequence*" — that is, a positionally ordered collection of other objects.
  - Strictly speaking, strings are sequences of one-character strings;
  - Maintain a left-to-right order
  - Stored and fetched by their relative position.
- Other types of sequences include Lists and tuples.

String Operations

```
>>> S = 'Spam'
>>> len(S)                  # Length
4
>>> S[0]                    # The first item in S, indexing by zero-based position
'S'
>>> S[1]                    # The second item from the left
'p'
```

```
>>> S[-1]                   # The last item in S
'm'
>>> S[len(S)-1]             # Negative indexing, the hard way
'm'
```

```
>>> S                       # A 4-character string
'Spam'
>>> S[1:3]                  # Slice of S from offsets 1 through 2 (not 3)
'pa'
```

```
>>> S[1:]                  # Everything past the first (1:len(S))
'pam'
>>> S                      # S itself hasn't changed
'Spam'
>>> S[0:3]                 # Everything but the last
'Spa'
>>> S[:3]                  # Same as S[0:3]
'Spa'
>>> S[:-1]                 # Everything but the last again, but simpler (0:-1)
'Spa'
>>> S[:]                   # All of S as a top-level copy (0:len(S))
'Spam'
```

```
>>> S
Spam'
>>> S + 'xyz'              # Concatenation

'Spamxyz'
>>> S                      # S is unchanged
'Spam'
>>> S * 8                  # Repetition
'SpamSpamSpamSpamSpamSpamSpamSpam'
```

Immutability

- Strings are *immutable* in Python—they cannot be changed in-place after they are created

```
>>> S
'Spam'
>>> S[0] = 'z'              # Immutable objects cannot be changed
...error text omitted...
TypeError: 'str' object does not support item assignment

>>> S = 'z' + S[1:]         # But we can run expressions to make new objects
>>> S
'zpam'
```

Strings

- String operations so far -> [:] are really generic sequence operations—these operations will work on other sequences in Python as well, including lists and tuples.

- Strings also have operations all their own, **available as *methods***.
- methods are specific functions attached to the object, which are triggered with a call expression like: `str.method()`.

How would you find all the methods available?

dir(st)

Use the dir function to get the method names.

```
>>> st='abcd'
>>> dir(st)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__
', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__sub
classhook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'fin
d', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', '
lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', '
rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', '
zfill']
```

Type-Specific Methods

```
>>> S.find('pa')              # Find the offset of a substring
1
>>> S
'Spam'
>>> S.replace('pa', 'XYZ')    # Replace occurrences of a substring with another
'SXYZm'
>>> S
'Spam'
```

```
>>> line = 'aaa,bbb,ccccc,dd'
>>> line.split(',')           # Split on a delimiter into a list of substrings
['aaa', 'bbb', 'ccccc', 'dd']
>>> S = 'spam'
>>> S.upper()                 # Upper- and lowercase conversions
'SPAM'

>>> S.isalpha()               # Content tests: isalpha, isdigit, etc.
True

>>> line = 'aaa,bbb,ccccc,dd\n'
>>> line = line.rstrip()      # Remove whitespace characters on the right side
>>> line
'aaa,bbb,ccccc,dd'
```

Method vs Function

- Though sequence operations are generic (i.e. str[3] or list[3]) methods are not
  - But some do share common method names.
- As a rule of thumb, Python's toolset is layered:
  - Generic operations that span multiple types show up as built-in functions or expressions (e.g., len(X), X[0]), but type-specific operations are method calls (e.g., str.upper()).
- Just takes some practice to become comfortable

help()

- Use the help function to get the method details.

```
>>> help(S.replace)
Help on built-in function replace:

replace(...)
    S.replace (old, new[, count]) -> str

    Return a copy of S with all occurrences of substring
    old replaced by new.  If the optional argument count is
    given, only the first count occurrences are replaced.
```

Quotes

- Double quotes mean the same thing as single
- Triple quotes allows multiline string literals
  - When this form is used, all the lines are concatenated together, and endof-line characters are added where line triple quotes closed.
  - Useful for embedding things like HTML and XML code in a Python script:

```
>>> S = 'A\nB\tC'                  # \n is end-of-line, \t is tab
>>> len(S)                        # Each stands for just one character
5

>>> ord('\n')                     # \n is a byte with the binary value 10 in ASCII
10

>>> S = 'A\0B\0C'                 # \0, a binary zero byte, does not terminate string
>>> len(S)
5
```

```
>>> msg = """ aaaaaaaaaaaa
bbb'''bbbbbbbbbb""bbbbbbb'bbbb
ccccccccccccc"""
>>> msg
'\naaaaaaaaaaaaa\nbbb\'\'\'bbbbbbbbbb""bbbbbbb\'bbbb\ncccccccccccccc'
```