assignment (=) vs equality (==)

```
x = 15
print(x)
print(x==15) # comparison
print(x==16) # comparison

# print(x=17)
# Python says: Above code is not valid.
# Reason: "x=17" is not an expression: would not produce resultant value for printing

# 1, -132, 3.14 are numeric values, True and False are Boolean values
x = 15
y = 16
a = x<y
b = x>y
c = x==y
d = x==(y-1)

# To beginners:
#       Don't Try print = 100, unless you will "restart kernel"
```

More:

- **Tuple assignment:** x, y, z = '15', '30', 15
- **Chained assignment:** x = y = z = 0
- **delete variable:** del x, y

Naming of identifiers:

identifier names: must start with letter or _
can follow by letter or digit or _

Data types:

No maximum for an integer, has maximum for float

Integer (the range is unlimited)

```
x = 10**90**2 # x = 10**90**? try 3, 4, 5
print(x/x)
print(x)

"there is no longer a limit to the value of integers. However .."
(https://docs.python.org/3.1/whatsnew/3.0.html#integers)
```

Floating Point Numbers (the range is limited)

```
%%ai
tell me the range of float in python in one line
```

Python float range: min ≈ 2.2250738585072014 e-308,
max ≈ 1.7976931348623157 e+308.

```
from sys import float_info
float_info
```

```
sys.float_info(max=1.7976931348623157e+308, max_exp=102
4, max_10_exp=308, min=2.2250738585072014e-308, min_exp=
-1021, min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.2
20446049250313e-16, radix=2, rounds=1)
```

```
x = -0.1          # decimal number
y = -1.0e-1     # scientific notation
z = -1 / 10       # fraction
a = 1234
x, y, z, a, type(x), type(y), type(z), type(a)

import sys
print(sys.float_info.max)
print(sys.float_info.max * 2)

x = sys.float_info.max
x1 = x - 2000000000000000000000000000000000000
x2 = x - 2e250
x3 = x - 2e307
x, x1, x2, x3, x==x1, x==x2, x==x3, x-x1, x-x2, x-x3
```

"I have $1234891234.234189"
=> (If only the 5 most significant digits are stored):
$1234800000.000000

Rounding a number

```
x = 28793.54836
print(round(x, 2))
print(round(x, 1))
print(round(x, 0))
print(round(x, -1))
print(round(x, -2))

round(150, -2), round(250, -2),round(350, -2), round(450, -2),round(550, -2), round(650, -2)
# The "Round to nearest, ties to even" rule
# https://en.wikipedia.org/w/index.php?title=IEEE_754#Rounding_rules

for i in range(10):
    x=i*100+50;
    print(x,"=>",round(x,-2))
```

String Formatting

```
hour, minute, second = 9, 10, 11
print("The time is {}:{}:{}".format(hour, minute, second))

hour, minute, second = 9, 10, 11
message = "The time is {}:{}:{}"
print(message.format(hour, minute, second))

hour, minute, second = 9, 10, 11
message = "The time is {:02d}:{:02d}:{:02d}"
print(message.format(hour, minute, second))
```

```
price = 913.5
message = "10% Discounted: {:.2f}, 20% Discounted: {:.2f}"
print(message.format(price*0.9, price*0.8))

# using indices (0-based)
hour, minute, second = 10, 9, 8
message = "The time is {0}:{1}:{2}. Second is {2}."
print(message.format(hour, minute, second))

# using names
print("The surname of {first} {last} is {last}.".format(first="John", last="Doe"))

# f-string, evaluate variables inside the replacement field
# Objective: show "202602 CS1302 Intro to Comp Prog'g"
y, m = 2026, 2 # or from user inputs
s1 = "{y}{m} CS1302 Intro to Comp Prog'g"
s2 = f"{y}{m} CS1302 Intro to Comp Prog'g"
s3 = f"{y}{m:02} CS1302 Intro to Comp Prog'g"

print(s3)
print(f"{y}{m:02} CS1302 Intro to Comp Prog'g")
print("{}{:02} CS1302 Intro to Comp Prog'g".format(y,m))
```

```
# Objective: show "202602 CS1302 Intro to Comp Prog'g"
y, m = 2026, 2
s1 = "{y}{m} CS1302 Intro to Comp Prog'g"
s2 = f"{y}{m} CS1302 Intro to Comp Prog'g"
s3 = f"{y}{m:02} CS1302 Intro to Comp Prog'g"

print(s3)
print(f"{y}{m:02} CS1302 Intro to Comp Prog'g")
print("{}{:02} CS1302 Intro to Comp Prog'g".format(y,m))
```

ne that just executed
ext line to execute

<< First | < Prev | Next > | Last >> | Permalink

Done running (7 steps)

Print output (drag lower right corner to resize)

```
202602 CS1302 Intro to Comp Prog'g
202602 CS1302 Intro to Comp Prog'g
202602 CS1302 Intro to Comp Prog'g
```

Frames

Global frame

y	2026
m	2
s1	"{y}{m} CS1302 Intro to Comp Prog'g"
s2	"20262 CS1302 Intro to Comp Prog'g"
s3	"202602 CS1302 Intro to Comp Prog'g"

Play with the following widget to learn more about the formating specification.

1. What happens when `align` is none but `fill` is `*`?
2. What happens when the `expression` is a multi-line string?

[4]:
```
@interact(
    expression=r"'ABC'",
    fill="*",
    align={"None": "", "<": "<", ">": ">", "=": "=", "^": "^"},
    width=(0, 20),
)
def print_object(expression, fill, align="^", width=10):
    format_spec = f"{{:{fill}{align}{'' if width==0 else width}}}"
    print("Format spec:", format_spec)
    print("Print:", format_spec.format(eval(expression)))
```

```
Last executed at 2026-01-20 17:23:45 in 19ms
```

expression 'ABC'

fill *

align <

width 10

```
Format spec: {:*<10}
Print: ABC*******
```

Error

3 kinds of program errors:

(1) Syntax error (cannot convert to runnable code to even start running)
(2) Runtime error (code stopped during running)
(3) Logical error (can finish, but wrong output)

Logical error

```
x = input("Enter a number: ")
y = input("Enter a number: ")
print("The sum is: ", x+y)
```

Syntax error vs Runtime error (e.g. a TypeError is detected at runtime)

```
print("Hello")
print("A+)
```

```
  Cell In[27], line 2
    print("A+)
          ^
SyntaxError: unterminated string literal
```

```
print("Hello")
print("100" + 3)
```

```
Hello
---------------------------------------------------------------
TypeError
Cell In[26], line 2
      1 print("Hello")
----> 2 print("100" + 3)

TypeError: can only concatenate str (not "int") to str
```

Operators (+,-,*,/, %, **, //),

binary *operators (+,-,*,/) have two operands: e.g. 3+4, 100/7*

unary *operators (-,+) has one operand: e.g. -123, +456*

//, %, **

```
# // is the integer division operator, return the quotient
# % is the modulo operator, return the remainder
100, 100/7, 100//7, 100%7 # divide 100 students into: 7 students per group
```

Operator overloading (+ and * are overloaded for strings)

```
s1 = 'abc' * 3
s2 = 10 * 'a'
s3 = "學"+"問"
s4 = "學" "問"
```

Statement VS Expression

* Given peter_mark = 40

"let me have the double of peter_mark" <== calculate it only (Peter doesn't feel happy)

"change peter_mark to its double" <== execute the change! (Peter is happy)

* statement: an instruction that Python executes

* expression: can be evaluated to give a resultant value

Examples: The following are statements.

What will they do when executed?

i) Assignment statement: x = x + 3

ii) Output statement: print("apple", x, 2*3)

Examples: The following are expressions.

What resultant values will be obtained?

iii) input("mark?") + " and " + input("bonus?")

iv) 2**10

v) 2500 + 100 / 7

vi) 1302

Do (i) and (ii) contain expressions? Does (iii) contain statements?

Assignment statement and Assignment Expression

* Try and explain: x = y = 1
(Answer: this is a special "chained assignment format")

Try and explain why it is invalid: x = (y = 1)
Try and explain why it is invalid: (x = y) = 1

* Python's assignment statement is not an expression: do not return any value

```
x = 13
print(x = 14) # error!!

x = 13
print(x == 15)
print(x := 16) # "Assignment Expression"
print(x)
```