

INIAD CS Essentials

2-1: Let's Use Functions

Using “Functions” in Python,

**you can do more complex calculations using abundant “vocabularies”
of Python**

1. Let's Make Functions

By making the program components called “functions”, you can efficiently program what you want

Let's convert Celsius(°C) to Fahrenheit(°F)

- In Japan, temperature is usually represented in Celsius (°C), while Fahrenheit(°F) is used in some countries
- According to Wikipedia, they can be converted using the following formula
 - $F = \frac{9}{5}C + 32$ (F: Fahrenheit, C: Celcius)
- What is 30 °C in Fahrenheit? Calculate using Python



30°C to Fahrenheit

- It is 86.0 °F, right?

$$\text{■} > 9 / 5 * 30 + 32$$

86.0

- Then, what about 10°C, 20°C, 40°C, 50°C, ...?

$$\text{■} > 9 / 5 * 10 + 32$$

$$\text{■} > 9 / 5 * 20 + 32$$

$$\text{■} > 9 / 5 * 40 + 32$$

$$\text{■} > 9 / 5 * 50 + 32$$

$$\text{■} > 9 / 5 * 60 + 32$$

$$\text{■} > \dots$$



The “DRY” Principle

- Computers are good at repeating the same things again and again, while humans are not
 - When humans do the same things over and over, they tend to be bored or be tired, which leads to the loss of their productivity while mistakes are increased
 - Did you notice that, in the previous slide, one equation was mistaken to put 「6」 in place where it has to be 「5」 ?
- The DRY (don't repeat yourself) Principle
 - Let's make it a habit to do any task only once, and let computers repeat the same tasks again and again
 - When you found yourselves repeating the similar things on computer, you need to be aware that you're wasting your time
 - When you found that, let's make it a habit to think, how can this situation be improved?

What you learn from here

- Let's make up a useful tool to avoid human repetition
- One you make up a component “f” that converts °C to °F, you do not need to write the similar expressions again and again

■ > f(10)
50.0

■ > f(20)
68.0

■ > f(30)
86.0

■ > f(-3.6)
25.52

■ > f(-100)
-148.0



Because I do not
have to write
expressions many
times, typing
mistakes can be
avoided!

Let's make the component "f"

- Let's input the following contents
(input all 3 lines in once)

```
> def f(t):  
    return 9 / 5 * t + 32  
>
```

At the end of first two lines,
press [Enter] to start
inputting the next lines

In the third(last) line, leave
the input blank and press
[Enter] to finish input

- If inputted correctly, the component is now available

Let's use the component you made

- To calculate using the component “f” you made, put a number you want to calculate in parentheses
- Try if your component works correctly

```
■ > f(20)
68.0
> f(30)
86.0
> f(25.3)
73.53999999999999
> f(-5.5)
22.1
```


Such component is called a “function”

- Remember the “functions” you learned in mathematics class at high school?
 - Linear and quadratic functions
 - Trigonometric functions (sin, cos, tan,...)
 - ...
- You can think of “function” as a component that, given a value (or values), corresponding result is outputted
 - Example: $\sin(30) = 0.5$ $\cos(45) = 0.7071072502...$
- Apparently, the component “f” you made is a function
 - $f(20) = 68.0$, $f(30) = 86.0$, ...

Definition of functions

- In Python, making function is called “function definition”
- Taking $f(t)$ you made like below as an example, we'll now check how we can define functions in Python

```
> def f(t):  
..     return 9 / 5 * t + 32
```

Reading the first line

- Function definition starts by writing

「**def** *function_name(variable1, variable2, ...)*:」

- Keyword “**def**” stands for definition; used to indicate that function definition from starts from here in Python
- For *variable1, variable2, ...*, list up the variable names for values passed to the function
- Don't forget the last colon 「**:**」!

```
> def f(t):  
..     return 9 / 5 * t + 32
```

Reading the first line (cont'd)

- In function definition of $f(t)$, what are the function name and variable names?

Function
(component)
name is "f"

Only one value is
passed to "f"; and
it's named "t"

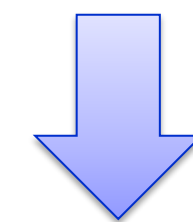
```
> def f(t):  
..     return 9 / 5 * t + 32
```

From the second lines, write what to do in function

- After started function definition by "**def**" in the first line, write the calculation process of that function from the second line
- To end the calculation in a function and return the calculation result, write 「**return expression**」

```
> def f(t):  
..     return 9 / 5 * t + 32
```

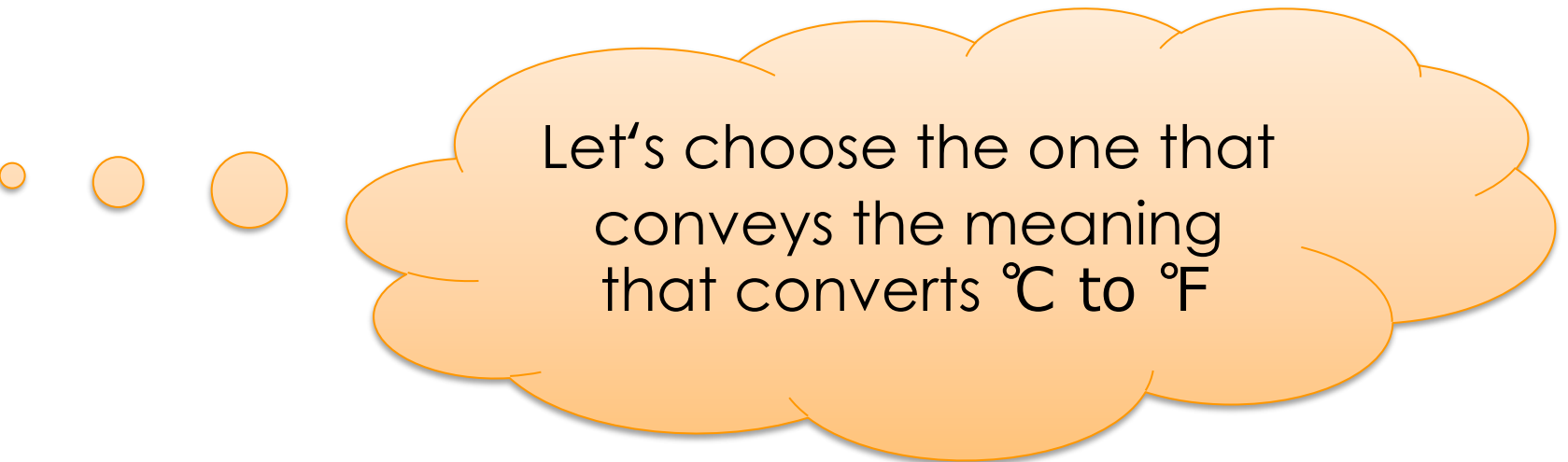
Ends calculation by returning
the result of $f(t)$ as
 $9 / 5 * t + 32$



From the second line,
write the calculation
process of the function

Like variables, function names may contain multiple characters

- Naming rules for functions are the same as those for variables
- As it is not restricted one character, try giving a good name easy to understand
- Question: If you want to make the name $f(t)$ easier to understand, which one do you think is the best?
 - `kashi_to_sesshi(ondo)`
 - `fahrenheit(temperature)`
 - `to_fahren(t)`
 - `C_to_F(temp)`
 - `Celcius_to_Fahrenheit(t)`
 - `convert_temperature(temp)`



Let's choose the one that
conveys the meaning
that converts °C to °F

Exercise 1: Kilometers(km) to miles(mile)

- Based on what you learned, let's actually define a function
- Define function `km_to_mile(d)` that converts distance in kilometers(km) to miles(mile)
 - Let us assume that $1[\text{mile}] = 1.609[\text{km}]$.
- When done, check if it works correctly
 - `> km_to_mile(1.609)`
`1.0`
 - `> km_to_mile(1)`
`0.6215040397762586`

Example answer

- You can define just like you did for $f(t)$

```
> def km_to_mile(d):  
..     return d / 1.609
```

Exercise 2: Polynomial computation

- Define function `polynomial(a, b, c, x)` that calculates the value of a polynomial $ax^2 + bx + c$
- Once done, check if it works correctly
 - `> polynomial(1, 2, 1, 3)`
16
 - `> polynomial(-1, -3, -2, 10)`
-72

Example answer #1

- This function accepts multiple values, but the idea is the same

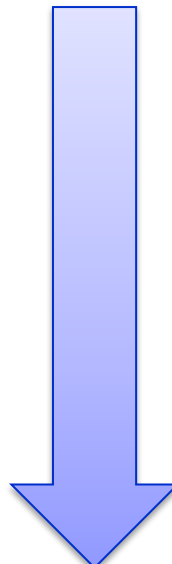
If you want to pass multiple values, you can list up the variable names separated by commas

```
> def polynomial(a, b, c, x):  
..     return a*x*x + b*x + c
```

Example answer #2

- You can divide the calculation process (starting from the second line) into multiple lines
- In that case, note that you need to align indentations and that the calculation is processed line by line from top

```
> def polynomial(a, b, c, x):  
..     first = a * x * x  
..     second = b * x  
..     third = c  
..     return first + second + third
```



Lines after the
second is executed
line by line from top

Function body is
aligned to the right
of "def"

When reached "return",
calculation is finished and
the result is returned

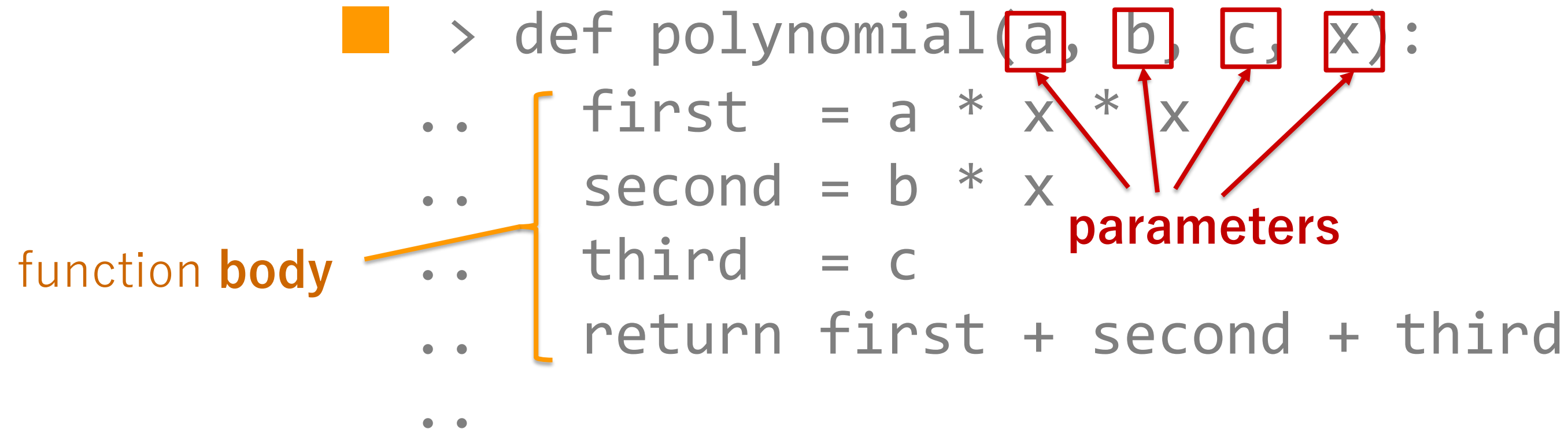
Let's learn some function-related terminologies

● Function definition

■ `> def polynomial(a, b, c, x):`
 `.. first = a * x * x`
 `.. second = b * x`
 `.. third = c`
 `.. return first + second + third`
 `..`

function **body**

parameters

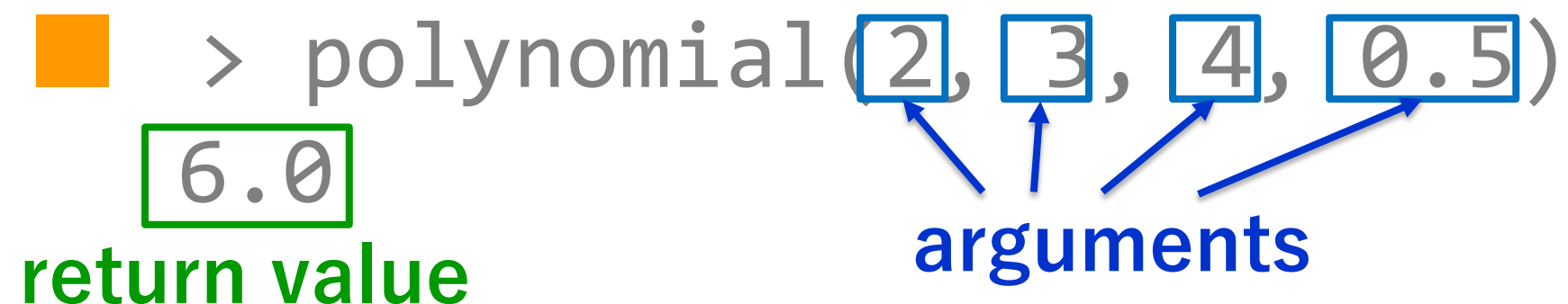


● Function call

■ `> polynomial(2, 3, 4, 0.5)`

`6.0`
return value

arguments



Exercise 3: Let's make a greeting message

- As you have learned, Python can handle not only numbers but also strings
- Define function `hello(name)` that, given `name` as an argument, and returns a greeting message in the form "Hello, ...!"
 - `> hello('Ken Sakamura')`
`'Hello, Ken Sakamura!'`
 - `> hello('Inoue Enryo')`
`'Hello, Inoue Enryo!'`

Example answer

- The idea remains the same, even when you are dealing with a string!
 - Do you remember what it means to “add” strings?

```
> def hello(name):  
..     return 'Hello, ' + name + '!'
```


2. Let's use built-in functions

You can use the functions provided by Python,
in addition to the functions that defined on your own

What are built-in functions?

- Let's recall type conversions learned last week

- `> int("123")`
123

- `> float("123.4") + 5.6`
129.0

- Built-in functions

- The functions provided from the start by Python
 - Actually, `int()`, `float()`, `str()` that you have learned earlier are all built-in functions

Examples of built-in functions (numbers)

● abs(x)

- Returns absolute value of x
- $\text{abs}(3.2) = 3.2$, $\text{abs}(-10.1) = 10.1$

● max(arg1, arg2, arg3, ...)

- Returns the maximum value among the values given as arguments
- $\text{max}(1, 2, 3, -1, -2) = 3$

● min(arg1, arg2, arg3, ...)

- Returns the minimum value among the values given as arguments
- $\text{min}(1, 2, 3, -1, -2) = -2$

● pow(x, y)

- Returns x to the power y (x^y)
- $\text{pow}(3, 4) = 81$, $\text{pow}(2, 0.5) = 1.4142135623730951$

● round(x), round(x, ndigits)

- Returns the rounded value of x
 - If ndigits is specified, rounded by leaving the number of digits after the dot
- $\text{round}(12.345) = 12$, $\text{round}(12.345, 2) = 12.34$

Examples of built-in functions (Other)

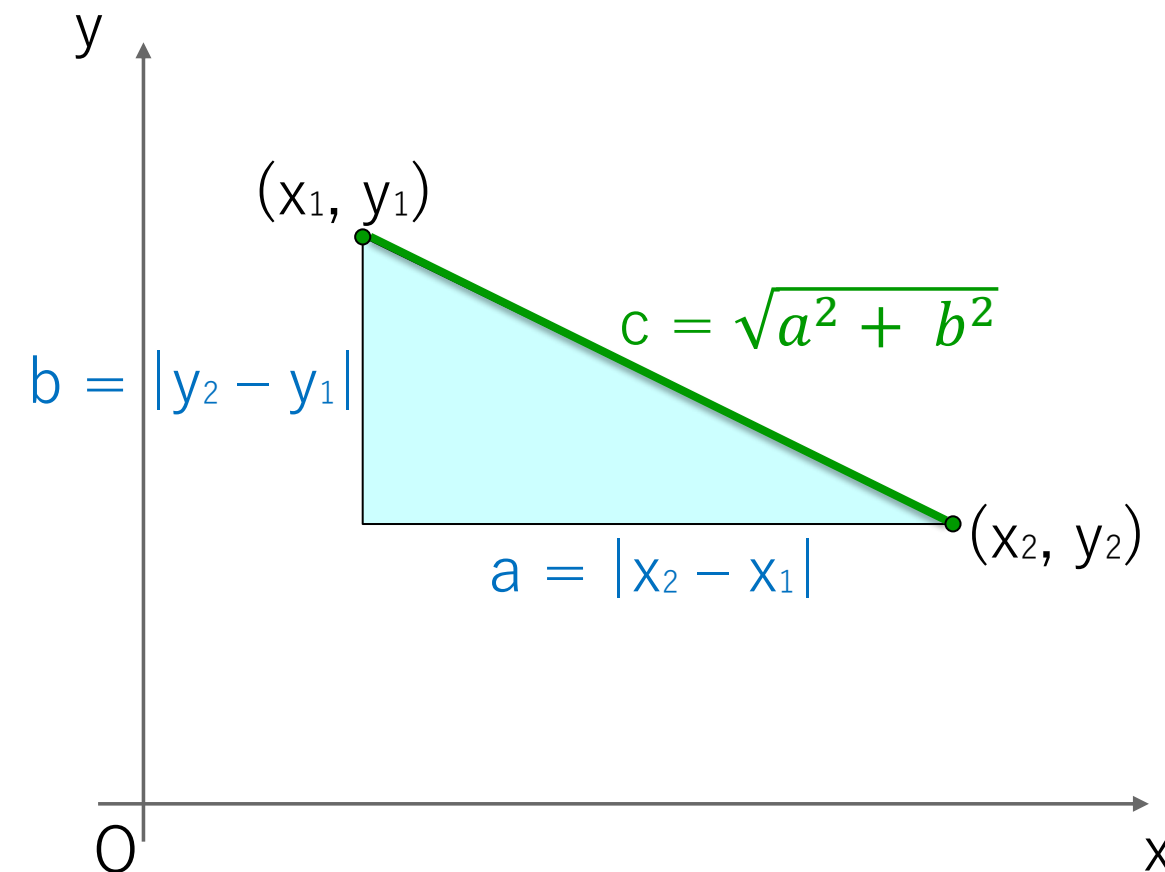
- `len(s)`
 - Returns the length of `s` (`s` can for example be strings)
 - `len("INIAD") = 5`, `len("情報連携") = 4`
- `type(x)`
 - Returns the type of `x`
 - `type("string") = <class 'str'>`,
`type(12.3) = <class 'float'>`
- `help()`, `help(x)`
 - Calls the help functions
 - If `x` is given, Python displays explanation of `x`
 - Example: `help(int)` : displays the manual of type `int`
(You can quit by typing 'q')

Exercise 4: Let's calculate the distance

- Define function `distance(x1, y1, x2, y2)` that calculates the distance of two points (x_1, y_1) , (x_2, y_2) on XY cartesian coordinates
- Hint
 - You remember Pythagorean theorem, right?
 - Recall the relationship of exponentials and radical roots you learned in high school. Square root can be calculated using built-in function `pow()`, right?
 - $\sqrt{x} = x^{1/2}$, $\sqrt[n]{x} = x^{1/n}$

Example answer #1: Simply written as it is

```
● > def distance(x1, y1, x2, y2):  
    ..     a = abs(x2 - x1)  
    ..     b = abs(y2 - y1)  
    ..     return pow(pow(a, 2) + pow(b, 2), 0.5)
```



Example answer #2: A little more efficient program

```
● > def distance(x1, y1, x2, y2):  
    ..     a = x2 - x1  
    ..     b = y2 - y1  
    ..     return pow(a * a + b * b, 0.5)
```



There are many
different ways to
define a same
function!

3. More operators

Let's learn to efficiently program
by learning several new operators

“Power operator” instead of `pow(x, y)`

- Power operator 「**」

- You can use `x ** y` and `pow(x, y)` to mean the same thing

- `> 2 ** 3`
8

- `> Pow (2, 3)`
8

Augmented assignment operator (composite operators)

- As introduced earlier, computer works by updating the “memory” contents
- Therefore, updating the value based on the current is a frequently-used process in programming
- Using augmented assignment operators ($+=$, $-=$, ...), you can write updating of variables concisely

All the left are equal to the right

You can read this as to
append a value

●> x += "aa"

●> x -= 1.0

●> x *= 2

●> x /= 5

●> x %= 3

●> x //= 10

●> x **= 4

●> x = x + "aa"

●> x = x - 1.0

●> x = x * 2

●> x = x / 5

●> x = x % 3

●> x = x // 10

●> x = x ** 4