

Flowchart Symbols

Flowchart Symbols

Name	Symbol	Description
Flow Line	→	Indicates the flow of logic by connecting symbols.
Terminal	○	Indicates the beginning and ending of Flowchart
Process	▭	Used to represent arithmetic operations and data-manipulations.
Decision	◇	Used to represent decision making between two or more alternatives.
Input/Output	▱	Used to represent input and output operation.
Predefined Process	▭	Shows named process which is defined elsewhere.
Connector	○	Used to join different flowline.

"Predefined Process" = Function

print Function & Formatting

Syntax	Explanation
<code>print('Hello Johnny.'</code>	Print string as output
<code>print('Hello\nJohnny.'</code>	Print "Johnny." on a new line
<code>print('Mon\tTues')</code>	Prints "Tues" at the next horizontal tab position (there's normally a tab position after every 8th character)
<code>print('One', 'Two', sep='~')</code>	Uses "~" (or other specified) as the item separator instead of a space Output: One~Two
<code>print('One', end=' ')</code> <code>print('Two')</code>	Prints a space instead of a newline character at the end of first output Output: One Two
<code>print(f'Hello {name}.')</code>	[F-string] Allows inserting variables & expressions.

--- Format Specifiers ---

print Function & Formatting (cont)

<code>print(f'{ payment:.2f}')</code>	[Precision designator] Value rounded to 2 decimal place output & displayed as floating point number.
<code>print(f'{ number:.2e}')</code>	Value is rounded to 2 decimal places in output & displayed in scientific notation.
<code>print(f'{ number:,}')</code>	[Comma separator] Display commas in large numbers ("10,000").
<code>print(f'{ dis count:.0%}')</code>	Converts decimal to percent output.
<code>print(f'{ number:10}')</code>	Sets minimum field width of
<code>print(f'{ number:^}')</code>	< = left align > = right align ^ = center align
<code>num = 12345.6789</code> <code>print(f'Num is {num:> 10,.2f}')</code>	Example with all specifiers. Output: Num is 12,345.68

Order of format specifiers:

`variable: [alignment] [width] [,] [.precision] [type]`

if Statements

```
if condition
```

Loops

```
#####
# while: for condition-controlled loops,
# & count-controlled loops (w/ added steps)
# Basic while loop:
while n < 5:
    print('Value is less than 5')
    n += 1
# Count-controlled while loop:
```



Loops (cont)

```
> n = 0 #Initialization
while n < 5: #Comparison
    print("Value is less than 5")
    n += 1 #Update
#####
#####
# for: for count-controlled loops & sequences of data
# (with or without "range")
for variable in [value1, value2, etc]:
    Statements
for variable in range(x, y, z):
    Statements #Range: x=start, y=end, z=step value
#####
#####
# else clause: (optional) for loops that
# contain a break statement
# Else clause never executes because break does execute:
for n in range(10):
    if n == 5:
        print("Breaking out of loop")
        break
    print(n)
else:
    print(f"After the loop, n is {n}.")
# Else clause does execute because break never does:
for n in range(3):
    if n == 5:
        print("Breaking out of loop")
        break
    print(n)
else:
    print(f"After the loop, n is {n}.")
```

Calculation Operators

<code>x [+ , - , *] y</code>	Addition, Subtraction, Multiplication
<code>x / y</code>	Floating-Point Division
<code>x // y</code>	Integer Division (positive results truncated; negative results rounded away from zero)
<code>x % y</code>	Remainder (returns remainder of $x \div y$)
<code>x ** y</code>	Exponent (returns value of x^y)

-- Special Operators --

Operator:	Equivalent To:
<code>x += y</code>	
<code>x -= y</code>	<code>x = x - y</code>
<code>x *= y</code>	<code>x = x * y</code>
<code>x /= y</code>	<code>x = x / y</code>
<code>x %= y</code>	<code>x = x % y</code>
<code>x //= y</code>	<code>x = x // y</code> (Note: This one not confirmed.)
<code>x **= y</code>	<code>x = x ** y</code> (Note: This one not confirmed.)

Boolean Expressions

<code>x > y</code>	x is greater than y
<code>x < y</code>	x is less than y
<code>x >= y</code>	x is greater than or equal to y
<code>x <= y</code>	x is less than or equal to y
<code>x == y</code>	x is equal to y
<code>x != y</code>	
<code>[expression1] and [expression2]</code>	both expressions must be true
<code>[expression1] or [expression2]</code>	at least one expression must be true
<code>not [expression]</code>	reverses value (false=true, true=false)



Variables

Assign and return variable's value: (walrus operator)

```
print(variable := value)
```

Set variable from input:

```
variable = input(prompt)
```

Set integer variable

```
variable = int(expression)
```

Set float variable: (decimal number)

```
variable = float(expression)
```

Set string variable from input:

```
variable = str(input(prompt))
```

Set Boolean variable: (true/false)

Single Line:

```
variable = val1 if condition else val2
```

Multiple Lines:

```
if condition:
    __ variable = val1
else:
    __ variable = val2
```

String concatenation:

```
message = 'Hello ' + 'world'
print(message)
```

Output: Hello world

Turtle Art!

Command	What It Does
<code>import turtle</code>	Necessary at start of program
<code>turtle.done()</code>	Put at very end of program (keeps window from auto-closing)

-- Colors & Sizes --

<code>turtle.setup(width, height)</code>	Set window size (in pixels)
<code>turtle.bgcolor('color')</code>	Set window background color
<code>turtle.pensize(width)</code>	Set pen thickness (in pixels)
<code>turtle.pencolor('color')</code>	Set pen color

-- Turtle Management --

Turtle Art! (cont)

`turtle.penup()` Raise pen (to move without drawing)

`turtle.pendown()` Lower pen (to resume drawing)

`turtle.hideturtle()` Hides turtle (does not affect drawing)

`turtle.showturtle()` Displays turtle

-- Input via Dialog Box --

`var = turtle.textinput('title', 'prompt')` Assigns user's input to the variable as a string

`var = turtle.numinput('title', 'prompt')` Assigns user's input to the variable as a float

`var = turtle.numinput('t', 'p', default=x, minval=y, maxval=z)` Optional arguments, `x` = default value displayed in input box, `y` = reject any number less than `z` = reject any number greater than `z`

-- Movement & Positioning --

`turtle.forward(n)` moves turtle `n` distance in the direction it's currently facing

`turtle.setheading(n)` Set turtle heading to a specific angle (90 is up)

`turtle.right(n)` Turn n degrees to the right

`turtle.left(n)` Turn n degrees to the left

`turtle.goto(x, y)` Move turtle to specific coordinates

`turtle.pos()` Displays turtle's current coordinates

--- Drawing & Writing ---

`turtle.fillcolor('color')` Set color for filling shape



By **River L.** (Tamaranth)
cheatography.com/tamaranth/

Not published yet.
Last updated 24th June, 2025.
Page 3 of 5.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Turtle Art! (cont)

`turtle.begin_fill()` Use before drawing shape to be filled

`turtle.end_fill()` Use after shape has been drawn

`turtle.circle(radius)` Draw a circle with specified radius
)

`turtle.dot()` Draw a simple dot at current location

`turtle.write(text)` Writes text, with lower left corner of
1st character at turtle's coordinates

-- Erasing & Undoing --

`turtle.clear()` Erases all drawing, but **doesn't** reset
turtle position, pen color, or
background color

`turtle.reset()` Clears/resets everything **except**
window background color

`turtle.clearscreen()` Clears/resets *everything*
()



By River L. (Tamaranth)
cheatography.com/tamaranth/

Not published yet.
Last updated 24th June, 2025.
Page 4 of 5.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>