

### 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

`num = 12345.6789`  
`print( f'Num is {num:> 10,.2f}' )`  
 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 &gt; y</code>	<code>x is greater than y</code>
<code>x &lt; y</code>	<code>x is less than y</code>
<code>x &gt;= y</code>	<code>x is greater than or equal to y</code>
<code>x &lt;= y</code>	<code>x is less than or equal to y</code>
<code>x == y</code>	<code>x is equal to y</code>
<code>x != y</code>	
<code>[expression1] and [expression2]</code>	<code>both expressions must be true</code>
<code>[expression1] or [expression2]</code>	<code>at least one expression must be true</code>
<code>not [expression]</code>	<code>reverses value (false=true, true=false)</code>



### 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

<code>turtle.setheading(n)</code>	Set turtle heading to a specific angle (90 is up)
<code>turtle.right(n)</code>	Turn <i>n</i> degrees to the right
<code>turtle.left(n)</code>	Turn <i>n</i> degrees to the left
<code>turtle.goto(x, y)</code>	Move turtle to specific coordinates
<code>turtle.pos()</code>	Displays turtle's current coordinates

--- Drawing & Writing ---



By **River L.** (Tamaranth)  
[cheatography.com/tamaranth/](http://cheatography.com/tamaranth/)

Not published yet.  
 Last updated 3rd July, 2025.  
 Page 3 of 5.

Sponsored by **CrosswordCheats.com**  
 Learn to solve cryptic crosswords!  
<http://crosswordcheats.com>

### Turtle Art! (cont)

<code>turtle.fillcolor('color')</code>	Set color for filling shape
<code>turtle.begin_fill()</code>	Use before drawing shape to be filled
<code>turtle.end_fill()</code>	Use after shape has been drawn
<code>turtle.circle(radius)</code>	Draw a circle with specified radius
<code>turtle.dot()</code>	Draw a simple dot at current location
<code>turtle.write(text)</code>	Writes text, with lower left corner of 1st character at turtle's coordinates

### --- Erasing & Undoing ---

<code>turtle.clear()</code>	Erases all drawing, but <b>doesn't</b> reset turtle position, pen color, or background color
<code>turtle.reset()</code>	Clears/resets everything <b>except</b> window background color
<code>turtle.clearscreen()</code>	Clears/resets <i>everything</i>

