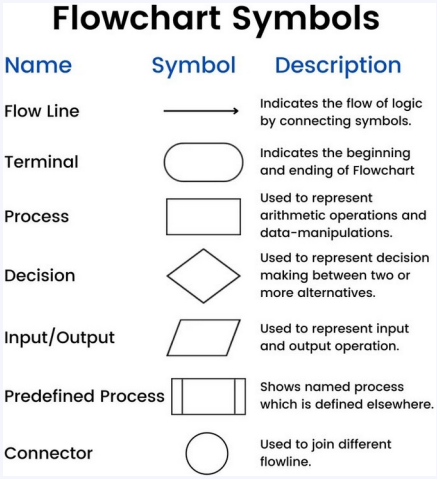
Cheatography

Python Basics Cheat Sheet

by River L. (Tamaranth) via cheatography.com/42882/cs/46556/

Flowchart Symbols



"Predefined Process" = Function

print Function & Formatting

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

--- Format Specifiers ---

print Function & Formatting (cont)

`print( f'{ payment:.2f}')`	[Precision designator] Valu rounded to 2 decimal place output & displayed as floati point number.
`print( f'{ number:.2e}')`	Value is rounded to 2 decim places in output & displaye scientific notation.
`print( f'{ number:,}')`	[Comma separator] Display commas in large numbers ("10,000").
`print( f'{ dis count:.0%}')`	Converts decimal to percen output.
`print( f'{ number:10}')`	Sets minimum field width o
`print( f'{ number:^}')`	< = left align > = right align ^ = center align
`num = 12345.6789` `print( f'Num is {num:> 10,.2f} ')`	Example with all specifiers. Output: `Num is  12,345`

Order of format specifiers:

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

if Statements

```
if condition
```

Loops

```
#####################################################
# while: for condit ion -co ntr olled loops,
# & count- con trolled loops (w/ added steps)
# Basic while loop:
while n < 5:
    print('Value is less than 5')
    n += 1
# Count- con trolled 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

`x [+, -, *] y`	Addition, Subtraction, Multiplication
`x / y`	Floating-Point Division
`x // y`	Integer Division (positive results truncated; negative results rounded away from zero)
`x % y`	Remainder (returns remainder of x ÷ y)
`x ** y`	Exponent (returns value of x^y)
	--- Special Operators ---
Operator:	**Equivalent To:**
`x += y`	
`x -= y`	x = x - y
`x *= y`	x = x * y
`x /= y`	x = x / y
`x %= y`	x = x % y

`x //= y`	x = x // y
	(**Note:** This one not confirmed.)

`x **= y`	x = x ** y (**Note:** This one not confirmed.)

Boolean Expressions

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

Cheatography

Python Basics Cheat Sheet

by River L. (Tamaranth) via cheatography.com/42882/cs/46556/

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 concat-enation:	`message = 'Hello ' + 'world'`
	`print( mes sage)`
	Output: `Hello world`

Turtle Art!

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

--- Colors & Sizes ---

`turtle.setup(width, height)`	Set window size (in pixels)
`turtle.bg color('color')`	Set window background color
`turtle.pe nsize(width)`	Set pen thickness (in pixels)
`turtle.pe nco lor('color')`	Set pen color

--- Turtle Management ---

Turtle Art! (cont)

`turtle.pe nup()`	Raise pe (to move without drawing)
`turtle.pe ndown()`	Lower pe (to resum drawing)
`turtle.hi det urtle()`	Hides tur (does no affect drawing)
`turtle.sh owt urtle()`	Displays turtle

--- Input via Dialog Box ---

`var = turtle.textinput ('title', 'prompt')`	Assigns user's input to tl variable a a string
`var = turtle.nu minput ('title', 'prompt')`	Assigns user's input to tl variable a a float
`var = turtle.nu minput ('t', 'p', default=x , minval=y, maxval=z)`	Optional argumen ***x*** = defau value displayec in input box ***y*** = rejec any number less than ***z*** = rejec any number greater than *z*

--- Movement & Positioning ---

`turtle.fo rward(n)`	moves turtle *n* distance direction it's currently facing

`turtle.se the ading(n)`	Set turtle heading t a specific angle (90 is up)
`turtle.right(n)`	Turn *n* degrees the right
`turtle.left(n)`	Turn *n* degrees the left
`turtle.goto(x, y)`	Move tur to specifi coordi-nates
`turtle.pos()`	Displays turtle's current coordi-nates
--- Drawing & Writing ---	
`turtle.fi llc olor('color')`	Set color for filling shape

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

Not published yet.
Last updated 24th June, 2025.

Turtle Art! (cont)	
`turtle.be gin _fill()`	Use before drawing shape to be filled
`turtle.en d_f ill()`	Use after shape has been drawn
`turtle.ci rcle(`*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.cl ear()`	Erases all drawing, but **doesn't** reset turtle position, pen color, or background color
`turtle.re set()`	Clears/resets everything **except** window background color
`turtle.cl ear scr een()`	Clears/resets *everything*