

Base Types			Container Types		
<code>integer, float, boolean, string</code>					
<code>int 783 0 -192</code>			<code>list [1, 5, 9] ["x", 11, 8.9] ["word"] []</code>		
<code>float 9.23 0.0 -1.7e-6</code>			<code>tuple (1, 5, 9) 11, "y", 7.4 ("word",) ()</code>		
<code>bool True False</code>	$10^{-6}$		<i>immutable</i>	<code>str as an ordered sequence of chars</code>	
<code>str "One\nTwo"</code>	<code>'I'm'</code>			<code>expression with just commas</code>	
<code>new line</code>	<code>'\n'</code>				
<code>multiline """X\ty\tz</code>	<code>1\t2\t3"""</code>				
<code>tab char</code>					
immutable, ordered sequence of chars					

Identifiers			Conversions		
<code>for variables, functions, modules, classes... names</code>			<code>type(expression)</code>		
<code>a..zA..Z_</code> followed by <code>a..zA..Z_0..9</code>			<code>int("15")</code>	can specify integer number base in 2 <sup>nd</sup> parameter	
▫ diacritics allowed but should be avoided			<code>int(15.56)</code>	truncate decimal part ( <code>round(15.56)</code> for rounded integer)	
▫ language keywords forbidden			<code>float("-11.24e8")</code>		
▫ lower/UPPER case discrimination			<code>str(78.3)</code>	and for literal representation <i>see other side for string formating allowing finer control</i>	<code>repr("Text")</code>
① <code>a toto x7 y_max BigOne</code>			<code>bool</code>	→ use comparators (with <code>==</code> , <code>!=</code> , <code>&lt;</code> , <code>&gt;</code> , ...), logical boolean result	
② <code>8y and</code>			<code>list("abc")</code>	use each element from sequence	<code>['a', 'b', 'c']</code>
			<code>dict([(3, "three"), (1, "one")])</code>		<code>{1: 'one', 3: 'three'}</code>
			<code>set(["one", "two"])</code>	use each element from sequence	<code>{'one', 'two'}</code>
			<code>":.".join(['toto', '12', 'pswd'])</code>	joining string	<code>'toto:12:pswd'</code>
			<code>"words with spaces".split()</code>	sequence of strings	<code>['words', 'with', 'spaces']</code>
			<code>"1,4,8,2".split(",")</code>	splitting string	<code>['1', '4', '8', '2']</code>

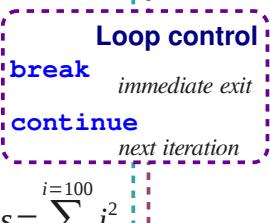
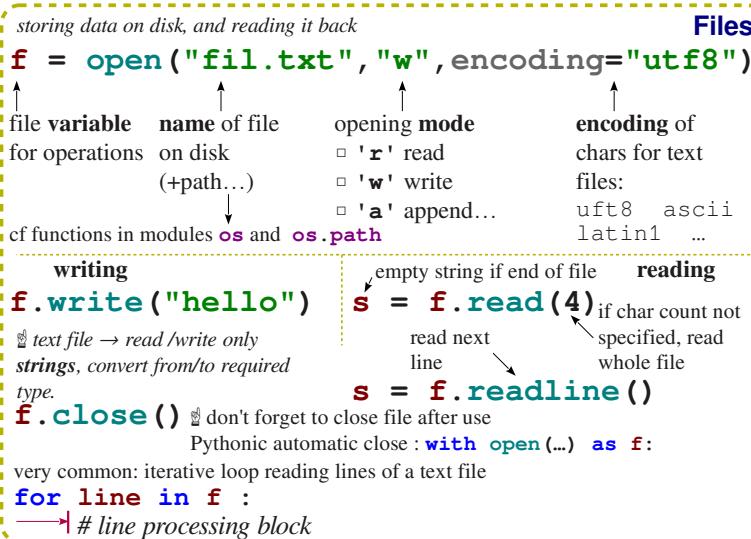
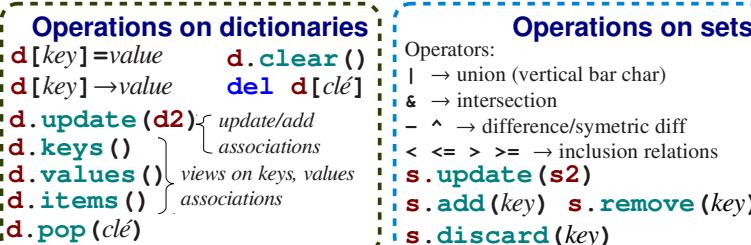
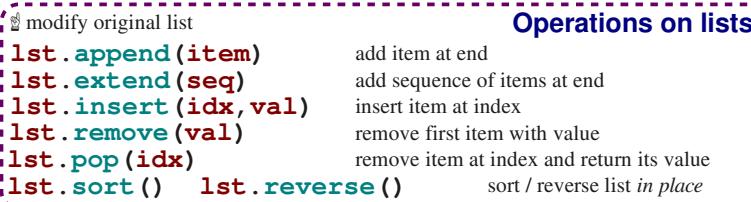
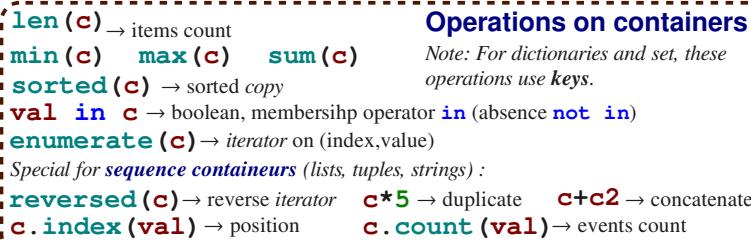
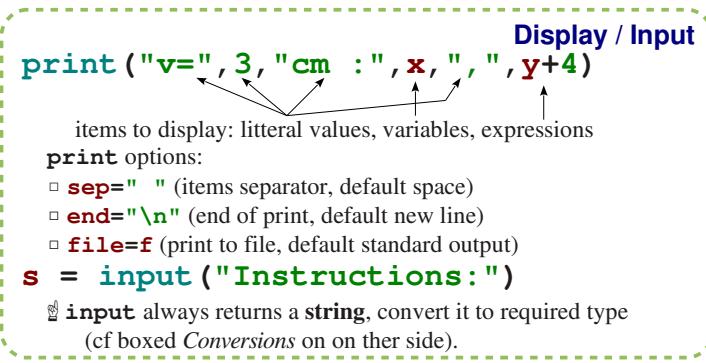
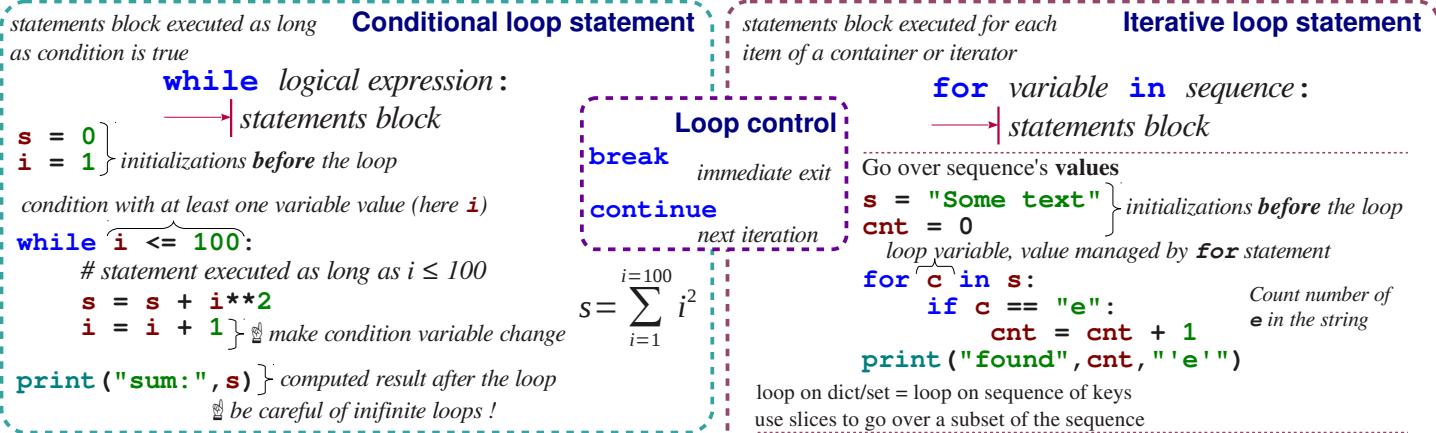
Variables assignment			Sequences indexing		
<code>x = 1.2+8+sin(0)</code>	value or computed expression		<code>for lists, tuples, strings, ...</code>		
variable name (identifier)			<code>len(lst) → 6</code>		
<code>y, z, r = 9.2, -7.6, "bad"</code>	variables names	container with several values (here a tuple)	individual access to items via <code>[index]</code>		
			<code>lst[1] → 67</code>	<code>lst[0] → 11</code>	first one
			<code>lst[-2] → 42</code>	<code>lst[-1] → 1968</code>	last one
			access to sub-sequences via <code>[start slice:end slice:step]</code>		
<code>x+=3</code>	increment		<code>lst[1:3] → [67, "abc"]</code>		
<code>x-=2</code>	decrement		<code>lst[-3:-1] → [3.14, 42]</code>		
<code>x=None</code>	« undefined » constant value		<code>lst[:3] → [11, 67, "abc"]</code>		
			<code>lst[4:] → [42, 1968]</code>		
			Missing slice indication → from start / up to end.		
			On mutable sequences, usable to remove <code>del lst[3:5]</code> and to modify with assignment <code>lst[1:4]=[ 'hop', 9]</code>		

Boolean Logic			Statements Blocks			Conditional Statement		
Comparators: <code>&lt; &gt; &lt;= &gt;= == !=</code>	<code>≤ ≥ = ≠</code>		<code>parent statement:</code>	<code>statements block 1...</code>	<code>:</code>	<code>statements block executed</code>		
<code>a and b</code>	logical and		<code>parent statement:</code>	<code>statements block 2...</code>	<code>⋮</code>	only if a condition is true		
<code>a or b</code>	both simultaneously		<code>indentation!</code>			<code>if logical expression:</code>		
<code>not a</code>	logical or					<code>statements block</code>		
<code>True</code>	one or other or both					can go with several elif, elif... and only one final else, example :		
<code>False</code>	logical not					<code>if x==42:</code>		
						<code># block if logical expression x==42 is true</code>		
						<code>print("real truth")</code>		
						<code>elif x&gt;0:</code>		
						<code># else block if logical expression x&gt;0 is true</code>		
						<code>print("be positive")</code>		
						<code>elif bFinished:</code>		
						<code># else block if boolean variable bFinished is true</code>		
						<code>print("how, finished")</code>		
						<code>else:</code>		
						<code># else block for other cases</code>		
						<code>print("when it's not")</code>		

Maths		
<code>floating point numbers... approximated values!</code>	angles in radians	
Operators: <code>+ - * / // % **</code>	<code>from math import sin, pi...</code>	
<code>x ÷ ↑ ↑ a<sup>b</sup></code>	<code>sin(pi/4) → 0.707...</code>	
integer ÷ remainder	<code>cos(2*pi/3) → -0.4999...</code>	
<code>(1+5.3)*2→12.6</code>	<code>acos(0.5)→1.0471...</code>	
<code>abs(-3.2)→3.2</code>	<code>sqrt(81)→9.0 √</code>	
<code>round(3.57, 1)→3.6</code>	<code>log(e**2)→2.0 etc. (cf doc)</code>	



$$s = \sum_{i=1}^{100} i^2$$

