
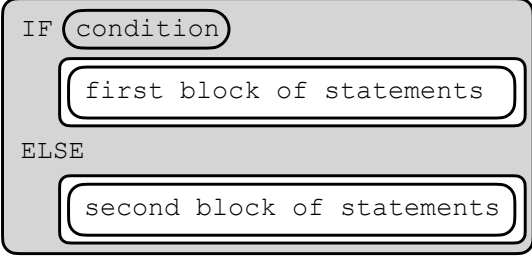
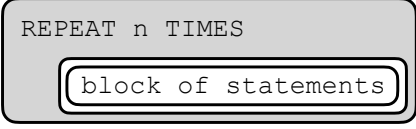

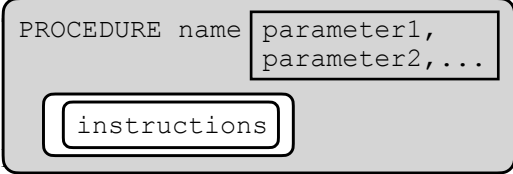
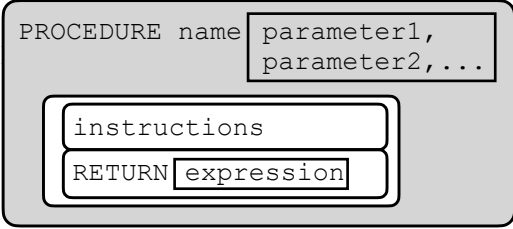


Swift AP Exam Language Reference Sheet

	Instruction (Swift)	Instruction (AP)	Explanation
Assignment, Display, and Input			
	Assignment <code>let a = expression</code> <code>var a = expression</code> <code>a = expression</code>  Examples <code>var name = "Douglas"</code> <code>let min = 0</code>	Text: <code>a ← expression</code>  Block: <div><code>a ← expression</code></div>	Evaluates <code>expression</code> and assigns the result to the variable <code>a</code> .
	Print Statements <code>print(expression)</code>  Examples <code>print("Hello, World!")</code> <code>print(min)</code>	Text: <code>DISPLAY (expression)</code>  Block: <div><code>DISPLAY</code> <code>expression</code></div>	Displays the value of <code>expression</code> , followed by a space.
	Because Swift handles user input differently, there is no direct parallel to <code>input()</code> .	Text: <code>INPUT ()</code>  Block: <code>INPUT</code>	Accepts a value from the user and returns it.
Arithmetic Operators and Numeric Procedures			
	Arithmetic Operators  <code>a + b</code> <code>a - b</code> <code>a * b</code> <code>a / b</code>	Text and Block:  <code>a + b</code> <code>a - b</code> <code>a * b</code> <code>a / b</code>	The arithmetic operators <code>+</code> , <code>-</code> , <code>*</code> , and <code>/</code> are used to perform arithmetic on <code>a</code> and <code>b</code> .  For example, <code>3 / 2</code> evaluates to <code>1.5</code> .
	Modulus (Remainder Operator) <code>a % b</code>  Example <code>17 % 4 == 1</code>	Text and Block:  <code>a MOD b</code>	Evaluates to the remainder when <code>a</code> is divided by <code>b</code> . Assume that <code>a</code> and <code>b</code> are positive integers.  For example, <code>17 MOD 5</code> evaluates to <code>2</code> .
	Random <code>Int.random(in: start...end)</code>  Example <code>Int.random(in: 0...100)</code>	Text: <code>RANDOM (a, b)</code>  Block: <code>RANDOM</code> <div><code>a, b</code></div>	Evaluates to a random integer from <code>a</code> to <code>b</code> , including <code>a</code> and <code>b</code> .  For example, <code>RANDOM (1, 3)</code> could evaluate to <code>1</code> , <code>2</code> , or <code>3</code> .
Relational and Boolean Operators			
	Relational Operators  <code>a == b</code> <code>a != b</code> <code>a &gt; b</code> <code>a &lt; b</code> <code>a &gt;= b</code> <code>a &lt;= b</code>	Text and Block:  <code>a = b</code> <code>a ≠ b</code> <code>a &gt; b</code> <code>a &lt; b</code> <code>a ≥ b</code> <code>a ≤ b</code>	The relational operators <code>=</code> , <code>≠</code> , <code>&gt;</code> , <code>&lt;</code> , <code>≥</code> , and <code>≤</code> are used to test the relationship between two variables, expressions, or values.  For example, <code>a = b</code> evaluates to <code>true</code> if <code>a</code> and <code>b</code> are equal; otherwise, it evaluates to <code>false</code> .
	Logical NOT Operator <code>!condition</code>  Example <code>let x = 4</code> <code>!(x &lt; 5) == false</code>	Text: <code>NOT condition</code>  Block: <code>NOT</code> <div><code>condition</code></div>	Evaluates to <code>true</code> if condition is <code>false</code> ; otherwise evaluates to <code>false</code> .

	Instruction (Swift)	Instruction (AP)	Explanation
	<p>Logical AND Operator</p> <p><code>&amp;&amp;</code></p> <p>Example</p> <pre>let x = -1 let y = 1 (x &lt; 0 &amp;&amp; y &gt; 0) == true (x &lt; 0 &amp;&amp; y &lt; 0) == false</pre>	<p>Text:</p> <p>condition1 AND condition2</p> <p>Block:</p> <p><code>condition1</code> OR <code>condition2</code></p>	<p>Evaluates to true if both condition1 and condition2 are true; otherwise, evaluates to false.</p>
	<p>Logical OR Operator</p> <p><code>  </code></p> <p>Example</p> <pre>let x = -1 let y = 1 (x &gt; 0    y &gt; 0) == true (x &lt; 0    y &gt; 0) == true (x &gt; 0    y &lt; 0) == false</pre>	<p>Text:</p> <p>condition1 OR condition2</p> <p>Block:</p>	<p>Evaluates to true if condition1 is true or if condition2 is true or if both condition1 and condition2 are true; otherwise, evaluates to false.</p>
Selection			
	<p>If Statement</p> <pre>if condition {     &lt;block of statements&gt; }</pre> <p>Example</p> <pre>// prints It's too hot! let temperature = 102 if temperature &gt; 99 {     print("It's too hot!") }</pre>	<p>Text:</p> <pre>{     &lt;block of statements&gt; }</pre> <p>Block:</p>	<p>The code in block of statements is executed if the Boolean expression condition evaluates to true; no action is taken if condition evaluates to false.</p>
	<p>If-Else Statement</p> <pre>if condition {     &lt;first block of statements&gt; } else {     &lt;second block of statements&gt; }</pre> <p>Example</p> <pre>// prints It's perfect! let temperature = 72 if temperature &gt; 99 {     print("It's too hot!") } else {     print("It's perfect!") }</pre>	<p>Text:</p> 	<p>The code in first block of statements is executed if the Boolean expression condition evaluates to true; otherwise, the code in second block of statements is executed.</p>
Iteration			
	<p>For Loop</p> <pre>for item in range {     &lt;block of statements&gt; }</pre> <p>Example</p> <pre>for number in 1...100 {     print(number) }</pre>	<p>Text:</p>  <p>Block:</p>	<p>The code in block of statements is executed n times.</p>
	<p>While Loop</p> <pre>while condition {     &lt;block of statements&gt; }</pre> <p>Note: Swift does not have a repeat until operation. A while loop can be used instead but the condition is the opposite of repeat until.</p> <p>Example</p> <pre>var n = 0 while n &lt; 4 {     print(n)     n += 1 }</pre> <p>is the same as:</p> <pre>REPEAT UNTIL (n &gt;= 4)</pre>	<p>Text:</p>  <p>Block:</p>	<p>The code in block of statements is repeated until the Boolean expression condition evaluates to true.</p>

	Instruction (Swift)	list <span>i</span> Instruction (AP)	Explanation
List Operations	In Swift, lists are zero-indexed, so the first element is at <code>list[0]</code> . If a Swift list index is less than 0 or greater than the length of the list minus 1, the program terminates with an error.	On the AP exam for all list operations, if a list index is less than 1 or greater than the length of the list, an error message is produced and the program terminates. There is no zero index in the AP language.	
	Accessing an Element <code>list[index]</code>  Example <code>let fruits = ["apple", "banana", "cherry"]</code> <code>print(fruits[0])</code> // prints apple <code>print(fruits[1])</code> // prints banana	Text: <div> <div>list[i]</div> ← <div>list[j]</div> </div> Block:  <div>list ← [value1, value2, value3]</div>	Refers to the element of <code>list</code> at index <code>i</code> . The first element of <code>list</code> is at index 1.
	Assigning a Value <code>list[i] = list[j]</code>  Example <code>var fruits = ["apple", "banana", "cherry"]</code> <code>let i = 1</code> <code>let j = 2</code> <code>fruits[i] = fruits[j]</code> // list is now "apple", "cherry", "cherry"	Text: <code>list[i]          list[j]</code> <div>list ← [value1, value2, value3]</div> Block:	Assigns the value of <code>list[j]</code> to <code>list[i]</code>
	Assigning Multiple Values <code>list = [value1, value2, value3]</code>  Example <code>var fruits = ["apple", "banana", "cherry"]</code> // list is now "apple", "cherry", "cherry"	Text:  Block: <div> <div>FOR EACH item IN list</div> <div>block of statements</div> </div>	Assigns <code>value1</code> , <code>value2</code> , and <code>value3</code> to <code>list[1]</code> , <code>list[2]</code> , and <code>list[3]</code> , respectively.
	For-Each Loop <code>for item in list {</code> <block of statements> <code>}</code>  Example <code>let numbers = [1,2,4,5,7,9]</code> // prints only even numbers <code>for number in numbers {</code> if number % 2 == 0 { print(number) } <code>}</code>	Text: FOR EACH item IN list { <block of statements> }  Block: <div>INSERT    list, i, value</div>	The variable <code>item</code> is assigned the value of each element of <code>list</code> sequentially, in order from the first element to the last element.  The code in <code>block of statements</code> is executed once for each assignment of <code>item</code> .
	Inserting a Value into a List <code>list.insert(value, at:index)</code>  Example <code>var fruits = ["apple", "banana", "cherry"]</code> <code>fruits.insert("grape", at: 1)</code> // list is now "apple", "grape", "banana", "cherry"	Text: INSERT (list, i, value)  Block: <div>APPEND    list, value</div>	Any values in <code>list</code> at indices greater than or equal to <code>i</code> are shifted to the right. The length of <code>list</code> is increased by 1, and <code>value</code> is placed at index <code>i</code> in <code>list</code> .
	Appending a Value to a List <code>list.append(value)</code>  Example <code>var fruits = ["apple", "banana", "cherry"]</code> <code>fruits.append("grape")</code> // list is now "apple", "banana", "cherry", "grape"	Text: APPEND (list, value)  Block: <div>REMOVE    list, i</div>	The length of <code>list</code> is increased by 1, and <code>value</code> is placed at the end of <code>list</code> .
	Removing a Value from a List <code>list.remove(at:index)</code>  Example <code>var fruits = ["apple", "banana", "cherry"]</code> <code>fruits.remove(at: 1)</code> // list is now "apple", "cherry"	Text: REMOVE (list i) LENGTH list  Block:	Removes the item at index <code>i</code> in <code>list</code> and shifts to the left any values at indices greater than <code>i</code> . The length of <code>list</code> is decreased by 1.

	Instruction (Swift)	Instruction (AP)	Explanation
	<p>Length of a List</p> <pre>list.count</pre> <p>Example</p> <pre>var fruits = ["apple", "banana", "cherry"] print(fruits.count)</pre>		Evaluates to the number of elements in list.
Procedures	In Swift, procedures are called functions. Functions associated with a type instance are called methods.		
	<p>Functions in Swift (No Return Value)</p> <pre>func name(label: Type) {     &lt;instructions&gt; }</pre> <p>Example</p> <pre>func greet(person:String) {     print("Hello, \(person)!") } greet(person:"Douglas") // prints Hello, Douglas!</pre>	<p>Text:</p> <pre>PROCEDURE name (parameter1,                   parameter2, ...)</pre> <pre>{     &lt;instructions&gt; }</pre> 	A procedure, name, takes zero or more parameters. The procedure contains programming instructions.
	<p>Functions in Swift (Return Value)</p> <pre>func name(label: Type) -&gt; Type {     &lt;instructions&gt;     return expression }</pre> <p>Example</p> <pre>func greet(person:String) -&gt; String {     return "Hello, \(person)!" } let greeting = greet(person:"Douglas") print(greeting) // prints Hello, Douglas!</pre>	<p>Text:</p> <pre>PROCEDURE name (parameter1,                   parameter2, ...)</pre> <pre>{     &lt;instructions&gt;     RETURN (expression) }</pre> <p>Block:</p>	A procedure, name, takes zero or more parameters. The procedure contains programming instructions and returns the value of expression. The RETURN statement may appear at any point inside the procedure and causes an immediate return from the procedure back to the calling program.