Python
Selective Processing

Selective Processing – If Statement

Perhaps the most well-known selective processing statement is the **if** statement. In its simplest form, the **if** statement is used to execute some code if a *condition* is true. For example:

Example:

```
x = int(input("Please enter an integer: "))
if x < 0:
    print ("The integer is a negative number.")
```

The **if** statement requires a condition that will evaluate to either ***True*** or ***False***. In the example above, the condition is ***x < 0***.

The condition is succeeded with a colon **:**. The statements that follow the colon are indented and are the statements that are executed if the condition is ***True***.

Conditional Operators

Operator	Description	**Example** Assume variable a holds 10 and variable b holds 20 then:
==	Checks if the value of two operands are equal or not; if values are equal then condition becomes true.	(a == b) is false.
!=	Checks if the value of two operands are equal or not; if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true. This is similar to != operator.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand; if yes then condition becomes true.	(a > b) is false.
<	Checks if the value of left operand is less than the value of right operand; if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes then condition becomes true.	(a >= b) is false.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes then condition becomes true.	(a <= b) is true.

Selective Processing – If..Else Statement

In its next simplest form, the **if..else** statement is used to execute some code if a *condition* is true and to execute some other code if the *condition* is false. For example:

Example:

```
#remember we must use int() around our input to convert the input
planets = int(input("How many planets orbit the star you discovered: "))
if planets > 0:
    print("How exciting, that star has it's own solar system")
else:
    print("That's too bad, the star is all alone")
```

The condition is succeeded with a colon **:**. The statements that follow the colon are indented and are the statements that are executed if the condition is ***True***. The statements that follow the **else:** are indented and are the statements that are executed if the condition is ***False***.

Ex. 1: (use one if..else statement) – write a program that asks the user for a fictional planet's distance from it's star in millons of Km. If the distance is less than 149 million km, the planet is closer to it's star than Earth is to the sun. Otherwise it's further away. Use an if and Else statement to determine the correct answer and tell the user if their fictional planet is closer to or further away than Earth is to the Sun.

Selective Processing – If..Elif..Else Statement

An **if..elif..else** statement allows only one of several options to be performed. The code after the first option whose condition is **True** is executed and the other code is not executed. If **none** of the conditions are **True** the code after the **else** is executed.

Example:

```
distance = int(input("How far is your planet from the sun? (in AU)"))
if distance < 1:
    print ("It's too hot on this planet to survive, you will burn.")
elif x > 1:
    print ("It's too cold on this planet to survive, you will Freeze")
else:
    print ("This planet is just right, It's in Goldilocks Zone")
```

There can be zero or more **elif** parts, but the last condition must always be **else**. The keyword 'elif' is short for 'else if', and is useful to avoid excessive indentation.

The importance of if… elif rather than if…if

When we write consecutive conditions using if and elif the conditions are linked and create a one or the other situation. This means as soon as one of the conditions are true the code will not even look at the subsequent conditions. This is important as it saves time for the computer and makes our code more efficient. Having multiple if statements it's possible for both if statements to be true at once which could give confusing output.

Ex. 2: (use one if..elif..else statement) – write a program that asks the user the age of the Sun in billions of years and tell the user which stage of it's life it's in:

0 to 10 billion years - Main Sequence

Display: The Sun is in its main sequence stage as it currently is now. In this stage the Sun uses nuclear fusion of hydrogen in its core to produce helium and emits energy as light and heat.

10 to 11 billion years – Red Giant

Display: The Sun is now a Red Giant. In this phase of its life is has used up all the hydrogen in its core and will now expand to 400 times its original size and engulf Earth completely. It will also cool and glow Red but we won't be around to see it.

More than 11 billion years – White Dwarf

Display: The sun is now a White Dwarf. This is the end stage for our Sun and it will shrink down to roughly the size of Earth. The Sun is not large enough to die in a supernova explosion as more massive stars do.

When checking the age of the sun we can say:

```
if age < 10:
        …
elif age < 11:
        …
```

For the elif we do not need to check if the age is greater than 10. If the age is less than 10 the first if statement would comeback as true and it would never check the second statement therefore if the code is looking at the elif statement we already know the age is greater than 10. This is more efficient than writing elif age > 10 and age < 11: