DATA TYPES AND OPERATORS

.

DATA TYPES

Data type states the way the values of that type are stored, the operations that can be done on that type, and the range for that type.

- ❑ Variables are used to store values of different types like character, integer and numbers with decimals etc. **(variable definition)**
- ❑ Different kinds of data require different amount of memory for storage.
- ❑ These different types of data can be manipulated through specific data types.

DIFFERENT TYPES OF DATA TYPES

There are four different types of data types:

❑ NUMERIC

- They are used to store only integer values.
- They can be byte(1 byte),short(2 bytes), int(4 bytes) and long(8 bytes)
- For eg. byte roll;

  int marks;

❑ FLOATING

- They are used to store numbers having decimal points
- They can be float(single precision: 4 bytes), double(double precision: 8 bytes)
- For eg. float percentage;

  double discount;

❑ CHARACTER

- They are used to store characters. Any symbol present on the keyboard is treated as character.
- We use char data type to store single value(enclosed in ' ') and string data type to store group of characters (enclosed in " ")
- For eg. char grade='A';

  String name=" Akshay";

❑ BOOLEAN

- They hold values either true or false
- They are 1 bit in size
- For eg. boolean b1=true;
  boolean b2=false;

VARIABLE NAMING CONVENTIONS

❑ Variable names are case sensitive. For eg. num and NUM are treated as two different variable names.

❑ Keywords or words, which have special meaning in java, should not be used as the variable names. For eg. class, main, public etc.

❑ Variable names should be short and meaningful.

❑ All variable names must begin with a letter, an underscore(_) or a dollar sign($).

❑ After the first initial letter, variable names may contain letters and digits (0 to 9) and (_,$), but no spaces or special characters are allowed.

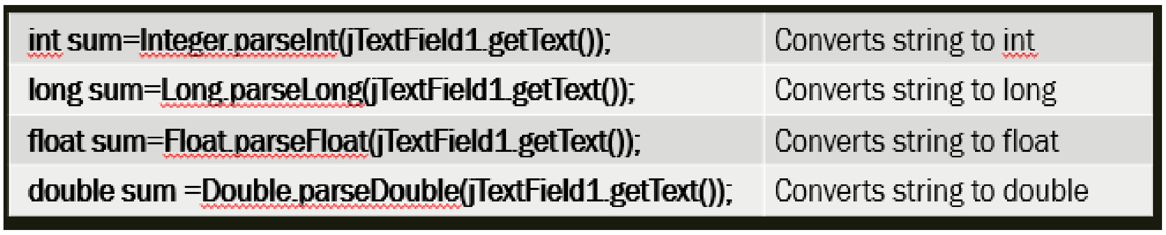
Examples of valid variable names: sum, marks1, first_name, $money

Examples of invalid variables names: marks%, 12grade, class, last-name

CONVERSION METHODS

1. CONVERTING STRING TO NUMBER:

We always get string value from text field. We can use parse method to convert it back to number.



int sum=Integer.parseInt(jTextField1.getText());	Converts string to int
long sum=Long.parseLong(jTextField1.getText());	Converts string to long
float sum=Float.parseFloat(jTextField1.getText());	Converts string to float
double sum =Double.parseDouble(jTextField1.getText());	Converts string to double

2. CONVERTING NUMBER TO STRING:

a. valueOf() method

b. toString() method

c. concatenation operator

```
int a=10;
float b=20.5;
double d=1000.678;
string s1=String.valueOf(a);
string s2=String.valueOf(b);
string s3=String.valueOf(d);
```

```
int a=10;
float b=20.5;
double d=1000.678;
string s1=Integer.toString(a);
string s2=Float.toString(b);
string s3=Double.toString(d);
```

```
int a=10;
float b=20.5;
double d=1000.678;
string s1=" "+ a;
string s2=b + "float variable";
//stores 20.5 float variable in
s2
string s3=" "+ d/2;
//stores  500.339 to s3
```

OPERATORS

Operators are symbols that manipulate, combine or compare variables.

- **Arithmetic**
  - **Unary** – They require a single operand to perform an operation. They can precede or follow their operands. For eg. unary +, unary -, increment/decrement operator.

OPERATOR	DESCRIPTION	EXAMPLE
UNARY -	To make the value of the number negative	-5
UNARY +	To make the value of the number positive	+5
INCREMENT ++	To increment the value by 1	Num++ or ++Num
DECREMENT --	To decrement the value by 1	Num—or --Num

  - **Binary** – They require two operands to perform an operation. For eg. +, -, *, /, % .

OPERATOR	DESCRIPTION	EXAMPLE
+	Add two operands	10+5=15
-	Subtracts two operands	10-5=5
*	Multiplies two operands	10*5=50
/	Divides two operands and gives quotient	**10/5=2** 10/3.0 = 3.3 10.0/3=3.3

		10/3=3
%	Divides two operands and gives remainder	10%5=0

- **Relational/Comparison Operator**: These operators describe the relationship between two operands. For eg. <,>,<=,>=,==,!=

OPERATOR	**DESCRIPTION**	**EXAMPLES**	
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	10<15, true 10>15, false	'rain'<'sun' true
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	15>10, true 15<10, false	'rain'>'sun' 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.	15<=15, true 13<=15, true 16<=15, false	'rain'<='sun' 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.	15>=15, true 16>=15, true 14>=15, false	'rain'>='sun' false
= =	Checks if the value of left operand is equal to the value of right operand, if yes then condition becomes true.	15= = 15, true 16 = = 15, false	'rain'=='sun' false
<>	Checks if the value of left operand is not equal to the value of right operand, if yes then condition becomes true.	15 <> 15, false 14 <>15, true	'rain'<>'sun' true

- **Assignment Operator**: Following assignment operators are supported by Python:

OPERATOR	**DESCRIPTION**	**EXAMPLE**

=	Assigns values from right side operands to left side operand	A = 10
+=	Adds right operand to the left operand and assign the result to left operand	A+=10 is equivalent to A =A +10
- =	Subtracts right operand from the left operand and assign the result to left operand	A-=10 is equivalent to A =A -10
*=	Multiply right operand by the left operand and assign the result to left operand	A*=10 is equivalent to A =A *10
/=	Divides left operand by the right operand and assign the quotient to left operand	A/=10 is equivalent to A =A /10
%=	Divides left operand by the right operand and assign the remainder to left operand	A%=10 is equivalent to A =A %10
**=	Performs exponential operations on operands and assigns the result to left operand	A **=2 is equivalent to A=A**2
//=	Performs floor division on operands and assigns the result to left operand	A//=2 is equivalent to A=A//2

- **Logical Operators**: Logical operators are used to join relational expressions.

OPERATOR	DESCRIPTION	EXAMPLE
not	NOT operator. It is a unary operator and it reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	>>> **not(5>10) = not(false)** **true**
or	OR Operator. It returns true if any of the conditions joined by this operator are true. It is false when both the conditions are false.	>>> **(10<5) or (15>20)** **false** >>> **(10>5) or (15>20)** **true** >>> **(10<5) or (15<20)** **true** >>> **(10<5) or (15< 20)** **true**

and	AND Operator. It returns true if both the conditions joined by this operator are true. It is false when any of the conditions are false.	>>> **(20>5) and (15>10)**
		true
		>>> **(20<5) and (15>10)**
		false
		>>>**(20>5) and (15<10)**
		false
		>>>**(20<5) and (15<10)**
		false

NOTE: Please remember

Expression	**Result**	**Expression**	**Result**
not(false)	**true**	**true and true**	**true**
not(true)	**false**	**false or true**	**true**
false and false	**false**	**false or false**	**false**
false and true	**false**	**true or false**	**true**
true and false	**false**	**true or true**	**true**

INCREMENT/ DECREMENT OPERATORS

Increment (++) operator and decrement (- -) operators are used to increase and decrease the value of variable respectively. There are two types of increment and decrement operators: pre and post.

- **PRE-INCREMENT/PRE-DECREMENT** - in a prefix expression, value is incremented/decremented first then this new value is restored back to the variable. For eg.
  Sum= ++num; // here first the value of num is incremented by 1 and then stored in sum.
- **POST-INCREMENT/POST-DECREMENT** -In postfix expression the current value is assigned to a variable then it is incremented/decremented by 1 and restored back to the original variable. For eg.
  Sum=num++; //here the value of num is first stored in sum and then it is incremented by 1.

For example: int num=10, sum=5;

++num; num++:	In both cases num will become 11
- num; num- -;	In both cases num will become 9.

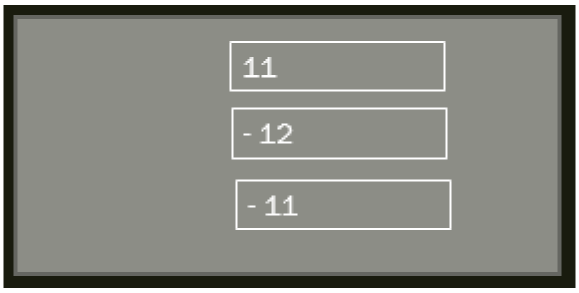
sum=++num + 5;	Here the value of num is first incremented and then added to 5 and stored in sum. Thus, sum=16 and num=11.
sum = num++ + 5;	Here the value of num is first added to 5 and stored in sum and then num is incremented by 1. Thus sum=15 and num=11.

OUTPUT QUESTION

Q Give the output of the following code:

```
int num=10;
 jtextfield1.settext(“ “+ num++);
num= -num;
jtextfield2.settext(“ “+ - - num);
num++;
jtextfield3.settext(“ “ + num);
```

OUTPUT



```
11
- 12
- 11
```

Explanation

int num=10;	variable num will be initialized by 10
jtextfield1.settext(“ “+ num++);	First the value of num is displayed in the text field and then it is incremented by 1 ie num becomes 11
num= -num;	Value of num is negated ie num becomes - 11
jtextfield2.settext(“ “+ - - num);	First the value of num is decreased by 1 ie num becomes – 12 and then it is displayed in the text field.
num++;	The value of num is incremented by 1 ie num becomes -11.
jtextfield3.settext(“ “ + num);	The value of num ie – 11 is displayed in the text field

IMPROTANT METHODS

1.setVisible(): To set the visibility of a component at run time.

setVisible(true) implies that the component is visible and setVisible(false) implies that the component is hidden.

SYNTAX

component.setVisible(boolean)

EXAMPLE

jLabel1.setVisible(true);	Makes the jLabel1 visible on the form
jLabel1.setVisible(false);	Makes the jLabel1 invisible on the form
jTextField1.setVisible(true);	Makes the jTextField1 visible on the form
jTextField1.setVisible(false);	Makes the jTextField1 invisible on the form
jButton1.setVisible(true);	Makes the jButton1 visible on the form
jButton1.setVisible(false);	Makes the jButton1 invisible on the form

2.setEditable(): To set the editing property of a component at run time.

setEditable(true) implies that the contents of this component can be changed at run time and setEditable(false) implies that the contents of this component cannot be changed at run time.

SYNTAX

component. setEditable(boolean)

EXAMPLE

jLabel1. setEditable(true);	Makes the jLabel1 editable on the form
jLabel1. setEditable(false);	Makes the jLabel1 ineditable on the form
jTextField1.setEditable(true);	Makes the jTextField1 editable on the form
jTextField1.setEditable(false);	Makes the jTextField1 ineditable on the form
jButton1.setEditable(true);	Makes the jButton1 editable on the form
jButton1.setEditable(false);	Makes the jButton1 ineditable on the form

3.setEnabled(): To set the enabled property of a component at run time.

The setEnabled(true) implies that this component can trigger a reaction at run time and setEnabled (false) implies that this component cannot trigger a reaction at run time.

SYNTAX

component. setEnabled(boolean)

EXAMPLE

jLabel1. setEnabled(true);	Makes the jLabel1 enabled on the form
jLabel1. setEnabled(false);	Makes the jLabel1 disable on the form
jTextField1.setEnabled(true);	Makes the jTextField1 enabled on the form
jTextField1.setEnabled(false);	Makes the jTextField1 disable on the form
jButton1.setEnabled(true);	Makes the jButton1 enabled on the form
jButton1.setEnabled(false);	Makes the jButton1 disable on the form