Session Quiz

Question -1

- Which of the following is not an identier
  - _____
  - ab_cd123
  - 1stName
  - abc_ABC

Question -2

- Which of the following is true about identifier name?

A. An identier can start with _, or <1-9>

B. An identifier can have multiple _ (underscores) in name

C. An identifier can have only lower or uppercase letters

D. An identifier can start only with a lower case letter

Question -3

- What will be the output

```
a, b, c = 10, 20, 30
a, b, c =  a+b, b+c, c+a
print (a, b, c)
```

A. 30, 50, 40

B. Invalid syntax

C. 10, 20, 30

D. 30, 20, 10

Question -4

- What will be the output

```
import math
a = 19.2
b = 20.6
print (ceil(a), floor(b))
```

A. 20 20

B. 19 20

C. 20 21

D. 20 19

Question - 5

- What will be the output

```
import math
a = “and”
b = “or”
print (a, b)
```

A. Syntax error. Use of reserbed words

B. and or

Question - 6

- What will be the output

```
import math
lambda = 19.5
pass = 50.2
print (ceil(lambda), floor(pass))
```

A. Syntax error. Use of reserved words

B. 20 50

C. 19 50

D. 20 51

Question - 7

- What will be the output

```
import math
a = 19.5
b = 50
c = "1"
print (type(a), type (b), type (c))
```

A. Syntax error. Use of reserved words

B. float int str

C. float int int

D. double int str

Question - 8

- Which of the following is true about variables

A. Variables store only the reference
B. Variables store the actual value
C. Variable values cannot be changed

Question - 7

- What will be the output

```
import math
a = 19.5
b = 50
c = "1"
print (type(a), type (b), type (c))
```

A. Syntax error. Use of reserved words

B. float int str

C. float int int

D. double int str

Data Types

Data Types

- The data stored in memory can be of many types.
  Example – student details
  - Name – Alphabets
  - Roll number – Alphanumeric
  - Age - Numeric
- Python provides various standard data types
  - Associated Operations
  - Storage method

Data Types

- 00000000 - 11111111
- 0 - 255
- -128 ....0....127

Numbers

- Number data types store numeric values.
- Number objects are created when a value is assigned
  - Var1 = 1 Var2 = 10
- del deletes the reference to a number object
  - del var1, var2…..
- a = 10
- print a
- del a
- print a // produces error as the reference is deleted

Importing Module – Computing n power m

- Open *PyCharm*
- Create new Python file using File->New->Python File -> number1.py
- Run->Run->number1.py

Numbers

- Python supports four different numerical types
  - int (signed integers)
  - long (long integers, they can also be represented in octal and hexadecimal)
  - float (floating point real values)
  - complex (complex numbers)

Numbers

- Open *PyCharm*
- Open file D:\Exercises\number2.py
- Click Run-.Run->*number2*
- What do you see as output?

Number

```
y = 100                    # x is integer number

print "Integer Number:", y

y = 3.14                   # x is float number

print "Float Number:", y

x = 100L                    # x is long number

print "Long Number:", x
```

Numbers – Further Practice

- Create new file : File->New-Python File->numberPractice.py
- Assign two integer,and two float variables with values and print them

Numbers – Complex Numbers

- Ordered pair of real floating-point numbers
- Denoted by **x + yj**
- **x** and **y** are real numbers
- **j** is the imaginary part (square root of -1)
- Example : 2+3j, 4-3j

Complex Numbers - Practice

- Open *PyCharm*
- Open file D:\Exercises\number3.py
- Click Run-.Run->*number3*
- What do you see as output?

Complex Number - Practice

```
x = 100 + 50j                 # x is complex number
y = 200 + 10j                 # y is another complex number

print "Complex Number x :", x
print "Complex Number y :", y
```

Numbers – Further Practice

- Create new file : File->New-Python File->numberPractice2.py
- Assign complex variables a and b and print them
- Define a third variable c
- c = a – b
- print a, b, c

Boolean

- Takes values *True* or *False*
- Open Boolean.py
- Run->Run->Boolean
- What output you see?

Boolean

- x = True
- y = False
- print "x = ", x
- print "y = ", y
- z = 1 > 2
- print "z = ", z
- print "1 < 2 is ", 1 < 2
- True
- False
- z = False
- 1 < 2 is True

Operators

Python Operators

- Python has the different operators which allows you to carry out required calculations in your program

Name	*Meaning*	*Example*	*Result*
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Float Division	1 / 2	0.5
//	Integer Division	1 // 2	0
**	Exponentiation	4 ** 0.5	2.0
%	Remainder	20 % 3	2

Operator

- Open *IDLE*
- Open file D:\Exercises\operator1.py
- Click Run->Run->*operator1*
- What do you see as output?

Operator – Further Practice

- Open file D:\Exercises\operator1.py
- Remove the assignment statements for x and y
- Add input statements for x and y
- Click Run->Run->*operator1*
- What do you see as output?

Operator

- Open *IDLE*
- Open file D:\Exercises\operator2.py
- Click Run->Run->*operator2*
- What do you see as output?

Complex Number Operators

- Addition of complex numbers x nd y
- x = a1 +b1j, y = a2 + b2j
- x + y = (a1+a2) + (b1 + b2)j
- Open *PyCharm*
- Open file D:\Exercises\complex_number.py
- Click Run->Run->*complex_number*
- What do you see as output?

Operator

```
x = 100 + 50j

print "Complex Number x:", x

y = 100 -25j

print "Complex Number y:", y

print "Addition of two Complex Number:", x+ y
```

Complex Number – Further Practice

- Open file D:\Exercises\complex_number.py
- Try other operations between complex numbers
- -, *, /
- Click Run->Run->*complex_number*
- What do you see as output?

Boolean Operator

- Comparison <, <=, >, >=, !=, ==
- Logical : and, or, not
- Load booleanOperator
- Click Run->Run->*booleanOperator*
- What do you see as output?

Boolean Operator

- x = True
- y = False
- z = True
- print "x = ", x
- print "y = ", y
- print "z = ", z
- x = True
- y = False
- z = True

- x = 1 < 2
- y = 1 > 2
- z = 1 == 2
- print "x = ", x
- print "y = ", y
- print "z = ", z
- x = True
- y = False
- z = False

- z = x and y
- w = x or y
- print "z = ", z
- print "w = ", w
- z = not x
- w = not y
- print "z = ", z
- print "w = ", w
- z = False
- w = True

Operator Precedence

- Expressions are evaluated using operator precedence
- The adjacent figure lists the python precedence order from high to low
- BODMAS
- Bracket Of Division
- Mult, Add, Sub

Operator	Description
`()`	Parentheses (grouping)
`f(args...)`	Function call
`x[index:index]`	Slicing
`x[index]`	Subscription
`x.attribute`	Attribute reference
`**`	Exponentiation
`~x`	Bitwise not
`+x, -x`	Positive, negative
`*, /, %`	Multiplication, division, remainder
`+, -`	Addition, subtraction
`<<, >>`	Bitwise shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`in, not in, is, is not, <, <=, >, >=, <>, !=, ==`	Comparisons, membership, identity
`not x`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`lambda`	Lambda expression

Operator Precedence

- 1 + 2 * 3+4
- 1*2 + 3*4

Operator Precedence

- Open *IDLE*
- Open file D:\Exercises\Operator_Precedence.py
- Click Run->Run->*Operator_Precedence*
- What do you see as output?

Operator Precedence

```
x = 2*(3 + 4)
y = 2*3 + 4

print "x: ", x
print "y: ", y

x = (-7 + 2)*(-4)
print "x: ", x

x = 4
y = 2
z = not 1 + 1 == y or x == 4 and 7 == 8

print "z: ", z
```

Operator Precedence – Further Practice

- Open file D:\Exercises\Operator_Precedence.py
- Add lines to print the following
- *2**4//6+2*
- *2**4//(6+2)*
- *2**(4+6)//2*
- *2**(4+6)//2*
- *not x << 2 & ~y*
- Click Run->Run->*Operator_Precedence*
- What do you see as output?

Importing Module – Further Practice

```
x = 2**4//6+2
y = 2**4//(6+2)

print "x: ", x
print "y: ", y

x = 2**4+6//2
y = 2**(4+6)//2
print "x: ", x
print "y: ", y

x = 8
y = 2
y = not x << 2  & ~y
print "x: ", x
print "y: ", y
```

Augmented Assignment Operator

- These operators allows you write shortcut assignment statements
- By using Augmented Assignment Operator we can write it as count = count + 1 as count += 1

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Float division assignment	`i /= 8`	`i = i / 8`
`//=`	Integer division assignment	`i //= 8`	`i = i // 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`
`**=`	Exponent assignment	`i **= 8`	`i = i ** 8`

Augmented Assignment Operator

- Open *IDLE*
- Open file D:\Exercises\Aug_Assign_Operator.py
- Click Run->Run->Aug_Assign_Operator
- What do you see as output?

Augmented Assignment Operator

```
x = 1
y = 2

print "x: ", x
print "y: ", y

x+= 10
print "x: ", x

x //=y
print "x: ", x

x **=y
print "x: ", x
```

Augmented Assignment – Further Practice

- File->New->Python File -> Aug_Operator2.py
- Add statements to
- input values into x and y
- print x /= y
- print x %/ y
- x */ y
- Click Run->Run 'Aug_Operator2.py'
- What do you see as output?

Augmented Assignment – Further Practice

```
x = 1
y = 2

print "x: ", x
print "y: ", y

x+= 10
print "x: ", x

x //=y
print "x: ", x

x **=y
print "x: ", x

x %= 10
print "x: ", x

x *=y
print "x: ", x

x /=y
print "x: ", x
```

Strings

Strings

- Strings are contiguous series of characters delimited by single or double quotes.
- Python don't have any separate data type for characters so they are represented as a single character string.
- Examples
- "abc", "my string", "Global Academy"

String Creation

- varName = “example” # string
- mychar = 'a’ # a character
- name1 = str() # this will create empty string object
- name2 = str("newstring") # string object containing 'newstring'

Strings

- Open *IDLE*
- Open file D:\Exercises\String.py
- Click Run->Run->String
- What do you see as output?

Strings

```
name = "tom" # a string
print(name)
mychar = 'a' # a character
print(mychar)
name1 = str() # this will create empty
string object
print(name1)
name2 = str("newstring") # string object
containing 'newstring'
print(name2)
```

Strings – Further Practice

- File->New->Python File->myString1.py
- Create new String with your Name
- Create empty String and modify it with your name
- Click Run->Run ‘myString1’
- What do you see as output?

Strings - Further Practice

```
name = "My Name is Manoj"
print(name)

myname = str() # this will create empty string object
print(myname)
myname = "This is Python Class"
print(myname)
```

Strings are immutable

- What this means to you is that once string is created it can't be modified.
- Every object in python is stored somewhere in memory. We can use id() to get that memory address
  - str1 = "welcome"
  - str2 = "welcome"
- Here str1 and str2 refers to the same string object "welcome" which is stored somewhere in memory

Strings are immutable

- Open *IDLE*
- Open file D:\Exercises\String_Immutable.py
- Click Run->Run->String_Immutable
- What do you see as output?

Strings are immutable

```
str1 = "welcome"
str2 = "welcome"

s1 = id(str1)
print(s1)

s2 = id(str2)
print(s2)

str1 += " mike"
print(str1)

s3 = id(str1)
print(s3)
```

Operations on Strings - indexing

- String index starts from 0
- str = “abc”
- str[0] =‘a’, str[1] = ‘’b’, str[2] = ‘c’
- open stringIndex1.py
- Run->stringIndex1.py
- Observe the output
- Add a line print str[3]
- Observe the output

Operations on Strings - indexing

- String index starts from 0
- str = “abc”
- str[0] =‘a’, str[1] = ‘’b’, str[2] = ‘c’
- open stringIndex.py
- Run->stringIndex.py
- Observe the output
- Add a line print str[3]
- Observe the output

Operations on Strings - subset

- Get subset of string from original string using [] operator (slicing operator)
  - Syntax: name[start:end]
- This will return part of the string starting from index start to index end - 1
- start and end are optional
-

Operations on Strings - subset

- Syntax: name[start:end]
- str = “abcde”
- str[0:2] = “ab”, str[1:3] = “bc”, str[:3] = “abc”
- str[1:] = “bcde”, str[:] = “abcde”
- Open stringIndex.py
- Run->stringIndex.py
- Add string variable to your program, Assign your full name to the string. Print only your first name and then the second name

Strings Operations

- “+” is used for concatenating strings
- str1 = “abc”, str2 = “def”, str1 + str2 = “abcdef”
- str1 = “abc”, str1 + “def” = “abcdef”
- f = “James”, s = “Bond”, Name = f + “ ” + n
- Name = “James Bond”
- * is used for repeating
- str1 = “abc”, str2 = str1 * 3 = “abcabcabc”

Strings Operations

- Open *IDLE*
- Open file D:\Exercises\String_Operation.py
- Click Run->Run->String_Operation
- What do you see as output?
- Define two new string variables, assign strings to them and print them concatenated

Operations on Strings

```
name = "tom"

print(name[0])

s = "tom and " + "jerry"
print(s)

s = "this is bad spam " * 3
print(s)
```

```
s = "Welcome"
s1 = s[1:3]
print(s1)

s2 = s[ : 6]
print(s2)

s3 = s[4 : ]
print(s3)

s4 = s[1 : -1]
print(s4)

s5 = s[:]
print(s5)
```

Strings Operations – Further Practice

- Open file D:\Exercises\String_Operation.py
- Create String with name “Hello World!”
- Print Hello World!
- Print H
- Print llo
- Print llo World!
- Print Hello World!Hello World!
- Print Hello World!TEST
- Run the file by clicking Run-> Run ‘String_Operation’

Operations on Strings – Further Practice

```
str = 'Hello World!'

print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

Strings Functions

Function Name	Function Description
len()	returns length of the string
max()	returns character having highest ASCII value
min()	returns character having lowest ASCII value
ord()	returns the ASCII code of the character
chr()	returns character represented by a ASCII number

Strings Functions

- str = “abcdefg”
- str.len() returns 7
- str.max returns ‘f’ (character with max ascii value)
- str.max returns ‘a’ (character with min ascii value)
- ord(str[0]) returns 97 (ascii value of a)
- chr(‘b’) returns 98

Strings Function

- Open *IDLE*
- Open file D:\Exercises\String_Function.py
- Click Run->Run->String_Function
- What do you see as output?

Strings Functions

```
s1 = len("hello")
print "Length of String hello:", s1

s2 = max("abc")
print"character having highest ASCII value in \"abc\":", s2

ch = 'b'
print "ASCII code of the character 'b':", ord(ch)

print "character represented by a ASCII number 97:", chr(97)
```

Strings Function

- Open *IDLE*
- Open file D:\Exercises\String_Function_1.py
- Click Run->Run->String_Function_1
- What do you see as output?

Strings Functions

```
s1 = "Manoj"
print "Length of String:", s1

s3 =  min(s1)
print "character having lowerst ASCII value in string:", s3

ch = 'A'
print "ASCII code of the character 'b':", ord(ch)

print "character represented by a ASCII number 97:", chr(90)
```

Strings Operations - Comparison

- Strings compares done using ASCII value of the characters
- Operators : > , < , <= , <= , == , !=)
- s1 = “abc”, s2 = “def” == > s1 < s2
- s1 = “Mary”, s2 = “Mac” == > s1 > s2
  - First two characters from M and M) are compared.
  - They are equal: second two characters are compared.
  - They are equal: third two characters (r and c) are compared.
  - Since 'r' has greater ASCII value than 'c' , str1 is greater than str2 .

Strings Comparison

- Open *IDLE*
- Open file D:\Exercises\String_Comparison.py
- Click Run->Run->String_Comparison
- What do you see as output?

Strings Comparison

```
print("tim" == "tie")
print("free" != "freedom")
print( "arrow" > "aron")
print("right" >= "left")
print("teeth" < "tee")
print("yellow" <= "fellow")
print("abc" > "")
str = “Global Academy
Print str[7:], “Academy”
```

Testing Strings

METHOD NAME	METHOD DESCRIPTION
isalnum()	Returns True if string is alphanumeric
isalpha()	Returns True if string contains only alphabets
isdigit()	Returns True if string contains only digits
isidentifier()	Return True is string is valid identifier
islower()	Returns True if string is in lowercase
isupper()	Returns True if string is in uppercase
isspace()	Returns True if string contains only whitespace

Strings Functions

- str = “abcdefg”
- str.isalnum() returns True
- str.isalpha() returns True
- str.isdigit() returns False
- str.islower() returns True
- str.isupper() returns False
- str.isspace() returns False

Strings Functions

- str = “abcd123”
- str.isalnum() returns True
- str.isaplha() returns False
- str.isdigit() returns False
- str.islower() returns True
- str.isupper() returns False
- str.isspace() returns False

Testing Strings

- Open *IDLE*
- Open file D:\Exercises\String_Testing.py
- Click Run->Run->String_Testing
- What do you see as output?

Testing Strings

```
s1 = "welcome to python"
s2 = "welcometopython"
print s1, " -  alphanumeric : ", s1.isalnum()
print s2, " -  alphanumeric : ", s2.isalnum()
print s1, " -  alphabets only : ", s1.isalnum()
print s2, " -  alphabets only : ", s2.isalnum()
s3 ="12345678"
print s1, " -  digits only : ", s1.isdigit()
print s3, " -  digits only : ", s3.isdigit()
s4 = "WELCOME"
print s2, " -  lowercase only : ", s2.islower()
print s4, " -  uppercase only : ", s4.isupper()
s5 ="     \t"
print s5, " -  spaces only : ", s5.isspace()
```

Testing Strings – Further Practice

- Open file D:\Exercises\String_Testing.py
- Modify strings s1, s2, s3, s4 and s5
- Run the file by Run->Run *python* String_Testing.*py*
- What do you see as output?

Searching for Substrings

METHOD NAME	METHOD DESCRIPTION
endswith(s1: str): bool	Returns True if strings ends with substring s1
startswith(s1: str): bool	Returns True if strings starts with substring s1
count(substring): int	Returns number of occurrences of substring the string
find(s1): int	Returns lowest index from where s1 starts in the string, if string not found returns -1
rfind(s1): int	Returns highest index from where s1 starts in the string, if string not found returns -1

Strings Functions

- str = “abcd123”
- str.endswith(“123”) returns True
- str.startswith(“abc”) returns True
- str = “abcd123abcd123”
- str.finds(“abc”) returns 0
- str.rfind(“123”) returns 11

Strings Search

- Open *IDLE*
- Open file D:\Exercises\String_Search_0.py
- Click Run->Run->String_Search_0
- What do you see as output?

Strings Search

```
str ="abcd123"
print str.endswith("123")
print str.startswith("abc")
str = "abcd123abcd123"
print str.count("abc")
print str.find("abc")
print str.rfind("123")
```

Strings Search

- Open *IDLE*
- Open file D:\Exercises\String_Search_1.py
- Click Run->Run->String_Search_1
- What do you see as output?

Searching for Substrings

```
s = "welcome to python"
s1 = s.endswith("thon")
print "String \"welcome to python\" ends with substring \"thon\":", (s1)

s2 = s.startswith("good")
print "String \"welcome to python\" starts with substring \"good\":", (s2)

s3 =  s.find("come")
print "lowest index from where substring \"come\" starts in \"welcome to python\":",(s3)

s4 = s.find("become")
print "lowest index from where substring \"become\" starts in \"welcome to python\":", (s4)

s5 = s.rfind("o")
print "highest index from where substring \"o\" starts in \"welcome to python\":", (s5)

s6 = s.count("o")
print "Number of occurrences of substring \"o\" in \"welcome to python\":", (s6)
```

Strings Search – Further Practice

- Open file D:\Exercises\String_Search_1.py
- Modify the strings to new values
- Run-> Run 'String_Search_1'
- What do you see as output?

Converting Strings

METHOD NAME	METHOD DESCRIPTION
capitalize(): str	Returns a copy of this string with only the first character capitalized
lower(): str	Return string by converting every character to lowercase
upper(): str	Return string by converting every character to uppercase
title(): str	This function return string by capitalizing first letter of every word in the string
swapcase(): str	Return a string in which the lowercase letter is converted to uppercase and uppercase to lowercase
replace(old,new): str	This function returns new string by replacing the occurrence of old string with new string

Strings Conversion

str =“new delhi"

print str.capitalize() prints New delhi

print str.upper() prints NEW DELHI

Print str.title() prints New Delhi

str = “Navi Mumbai”

print str.lower() prints navi mumbai

print str.swapcase() prints nAVI mUMBAI

print str.replace(“Navi”, “New”) prints New Mumbai

Converting Strings – Practice - 1

- Open *IDLE*
- Open file D:\Exercises\String_Converting_0.py
- Click Run->Run->String_Converting_0
- What do you see as output?

Converting Strings – Practice 2

- Open *IDLE*
- Open file D:\Exercises\String_Converting.py
- Click Run->Run->String_Converting
- What do you see as output?

Converting Strings

```
s = "string in python"
s1 = s.capitalize()
print(s1)

s2 = s.title()
print(s2)

s = "This Is Test"
s3 = s.lower()
print(s3)

s4 = s.upper()
print(s4)

s5 = s.swapcase()
print(s5)

s6 = s.replace("Is", "Was")
print(s6)
```

Converting Strings – Further Practice

- Create a Python program that does the followin
- Capitalizing first letter of String "*this is python workshop*"
- Capitalizing first letter of every word of String "*hello world*"
- Convert "*hello world*" into uppercase
- Convert "*Python WorkShop*" into lowercase

String Search – Further Practice

```
s = "this is python workshop"
s1 = s.capitalize()
print(s1)

s = "hello world"
s2 = s.title()
print(s2)

s = "Python WorkShop"
s3 = s.lower()
print(s3)

s = "hello world"
s4 = s.upper()
print(s4)
```

Thank You