Python Bootcamp & Masterclass

Identifiers & Variables

gknxt

objects

- Python is an object-oriented programming language. Objects are the core things that Python programs manipulate. All data in a Python program is represented by objects or by relations between objects.

- Every object in Python is assigned a unique identity (ID) which remains the same for the lifetime of that object. The built-in function `id( )` returns the identity of an object.

- The built-in function `type( )` returns the type of an object.

identifier

An identifier is just a valid name that is a nonempty sequence of characters of any length that consists of a “start character” and zero or more “continuation characters” that adheres to the following rules.

- The start character should be a Unicode letter or an underscore (“_”)
- Each continuation character can be a Unicode letter or an underscore (“_”) or a digit
- Unicode letters consist of alphabets of non-English languages, but non-English alphabet should be avoided unless its usage is a necessity for customization or localization

Avoid using the names of any of Python's predefined identifiers, built-in data types (int, float, list, str, tuple etc.) and built-in functions or exceptions.

```
print(dir())
```

```
['In', 'Out', '_', '_1', '_10', '_12', '_13', '_2', '_3', '_4', '_5', '_8', '__', '___', '__builtin__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', '_dh', '_i', '_i1', '_i10', '_i11', '_i12', '_i13', '_i14', '_i15', '_i16', '_i17', '_i2', '_i3', '_i4', '_i5', '_i6', '_i7', '_i8', '_i9', '_ih', '_ii', '_iii', '_oh', 'exit', 'get_ipython', 'list1', 'quit']
```

```
print(dir(__builtin__))
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__IPYTHON__', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate', 'eval', 'exec', 'filter', 'float', 'format', 'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

1. amount$
2. pass
3. 3rd_Qtr
4. _make
5. String
6. interest-rate
7. Ratio2k
8. yield
9. int
10. l_impôt4

identifier?
or not?

01 Identifiers are case-sensitive & cannot be python keywords

02 Identifiers can be made of letters, digits and underscores, except for the first character

03 Identifiers has to start with a letter or an underscore

04 Identifiers can be of any length

```
from keyword import iskeyword
print("amount$ a valid identifier?", 'amount$'.isidentifier() and not iskeyword('amount$'))
print("pass a valid identifier?", 'pass'.isidentifier() and not iskeyword('pass'))
print("3rdQtr a valid identifier?", '3rdQtr'.isidentifier() and not iskeyword('3rdQtr'))
print("_make a valid identifier?", '_make'.isidentifier() and not iskeyword('_make'))
print("String a valid identifier?", 'String'.isidentifier() and not iskeyword('String'))
print("interest-rate a valid identifier?", 'interest-rate'.isidentifier() and not iskeyword('interest-rate'))
print("Ratio2k a valid identifier?", 'Ratio2k'.isidentifier() and not iskeyword('Ratio2k'))
print("yield a valid identifier?", 'yield'.isidentifier() and not iskeyword('yield'))
print("int a valid identifier?", 'int'.isidentifier() and not iskeyword('int'))
print("l_impôt4 a valid identifier?", 'l_impôt4'.isidentifier() and not iskeyword('l_impôt4'))
```

```
amount$ a valid identifier? False
pass a valid identifier? False
3rdQtr a valid identifier? False
_make a valid identifier? True
String a valid identifier? True
interest-rate a valid identifier? False
Ratio2k a valid identifier? True
yield a valid identifier? False
int a valid identifier? True
l_impôt4 a valid identifier? True
```

1. amount$ ✗
2. pass ✗
3. 3rd_Qtr ✗
4. _make ✓
5. String ✓
6. interest-rate ✗
7. Ratio2k ✓
8. yield ✗
9. int ✓
10. l_impôt4 ✓

identifier? ✓
or not? ✗

variable

- A variable in a program is uniquely identified by a valid, unique name (identifier).
- A variable in Python refers to an object that is stored in the memory, so variable is just “a name given to a memory location” (a reference or pointer to an object)
- Variable declaration is implicit in Python (Variables are created automatically when they’re first assigned a value)
- Variables must be created before they are used

- It is conventional to use identifiers with lower case letters and underscores as variables names.

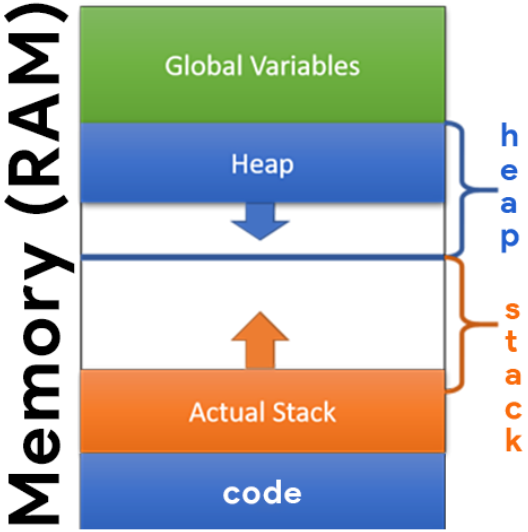
- Python does not have the concept of constants, so it is conventional to use identifiers with UPPER case letters and underscores to represent constants.

- In Python variables are just names (or labels or tags) that refer to objects in the Python interpreter's namespace.

- Any number of variables can refer to the same object, and when that object changes, the value referred to by all of those variables also changes.

Memory Management

While running a python program, computer memory gets divided into different sections:



- Global/Static Variables
- Heap
- Stack
- Code

Code section of the memory stores code in the machine-readable form. Python interpreter loads functions and local variables in the Stack, Global Variables in the Global Variables segment of the memory, and objects into the heap section of the memory

- Python interpreter actively allocates and deallocates the memory on the Heap to store/destroy objects and uses a garbage collection algorithm (called Garbage Collector) that keeps the Heap memory tidy as it removes objects that are not needed anymore or went out of scope

- Python interpreter load functions and local variables (references only) in the Stack. Stack memory is static (the size of values stored in the Stack cannot be changed) and temporary (as soon as the called function returned its value, the function and the related variable will be removed from the Stack). Python interpreter and OS memory management together keeps stack tidy.

Let's create a variable to store price

1. To create a variable, we need a valid name (identifier). Let's call our variable `price`
2. Variables are created automatically when they're first assigned a value. Python uses = as the assignment operator. Let's assign 2000 to `price`
3. Python creates an int object (since 2000 is an integer) stores it on heap and gives the address of that storage location to `price`

```
price = 2000        # variable price was created
```

```
type(price)         # int object with 2000 as value was created on heap
```

```
int
```

```
hex(id(price))      # memory address of that int object will be given to price
```

```
'0x20a91849230'
```

Disassembler

We can see how python bytecode is disassembled by importing dis module

`dis` — Disassembler for Python bytecode

Source code: Lib/dis.py

The `dis` module supports the analysis of CPython bytecode by disassembling it. The CPython bytecode which this module takes as an input is defined in the file `Include/opcode.h` and used by the compiler and the interpreter.

CPython implementation detail: Bytecode is an implementation detail of the CPython interpreter. No guarantees are made that bytecode will not be added, removed, or changed between versions of Python. Use of this module should not be considered to work across Python VMs or Python releases.

Online Resources

For best python resources, please visit:

gknxt.com/python/

gknxt

Python
Bootcamp
& Masterclass

Thank You

for your Rating & Review

gknxt