Types in Python

Thursday, 13 January 2022 00:03

• Types — Every value in Python has a particular type, and the types of values determine how they behave.

Definition: A type is a set of values and the operations that can be applied to those values

int : integer
float : floating point (decimal point between digits)

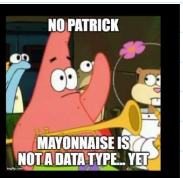
int ±/ int	→	int
float ±/ float	→	float
int ±/ float	→	float

(17. = 17.0)

string : – sequence of character
– Start and end with single quotes (') or double quotes (")
– Must be consistent

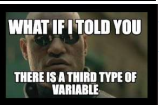
ex
```
>>> "How are you?"
>>> '"Hello haha" is so weird'
>>> "Hihi hee'
Syntax Error
```

Variable Types



- A **type** is a set of *values* and the *operations* that can be performed on those values.
- *int:* integer
  - ex. 3, 4, 894, 0, -3, -18
- *float:* floating point number
  - ex. -5.6, 7.342, 53452.0, -89.34

Type: str (pronounced string)



- *str:* string literal is a sequence of characters
- Start and end with single quotes (') or double quotes (")
  - ex. 'hello', "What is 10 * (2 + 9)?"
  - Just like writing in English, the quote type must match (i.e. 2 singles or 2 doubles, not 1 of each)

String Type Examples

```
>>> 'how are you?'

>>> "short- and long-term"

>>> '"APS106" is already my favourite course'

>>> "APS106 stinks'
SyntaxError
```

Variables and Memory

Thursday, 13 January 2022 15:29

1. Variables

Assigning values to variables → Use "="

variable = expression

Rules for assignment:

1. Evaluate the expression to the right of = sign (produces memory address of the value
2. Store the memory address in the variable of the left of the = sign

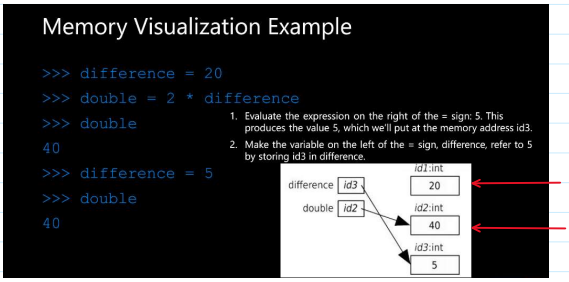
Variable legal names : MUST START WITH LETTER or _ (underscore)
CANNOT START WITH NUMBERS

2. Memory

Are "holes" with address to store the value

NOTE Python will create new id to store new variables instead of overwritting

ex
```
>>> a = 2
>>> b = 2*a        → a = 3
>>> a = 3           b = 4 (not 6)
```



id1 still exists, but not assigned

"double" still points to 40

- id(•) = Read the position of the variable in the memory (memory address)
  - = variable name, or value assigned to variable

ex
```
>>> x = 812
>>> id(x)
    781023568
>>> id(812)
    781023568
```

☐ Arithmetic in Python (+-*/)

Wednesday, 12 January 2022 23:34

• Arithmetic operations

Operator	Operation	Expression	English description	Result
+	addition	11 + 56	11 plus 56	67
-	subtraction	23 - 52	23 minus 52	-29
*	multiplication	4 * 5	4 multiplied by 5	20
**	exponentiation	2 ** 5	2 to the power of 5	32
/	division	9 / 2	9 divided by 2	4.5
//	integer division	9 // 2	9 divided by 2	4
%	modulo (remainder)	9 % 2	9 mod 2	1

• Integer Division, Modulo, and Exponentiation

Integer Division // : Takes the floor of division result (rounds down to nearest integer)

17 // 10 → 1

Modulo % : Remainder of division, matching the sign of second operand

53 % 24 → 5

NOTE : Python takes floor result, meaning rounding down, which also apply to negative numbers.

-17 // 10 → -2

17 % -10 → -3

division / always return the result as a float.

• Arithmetic Operator Precedence

Operator	Precedence
**	highest
- (negation) → Makes a number negative	
*, /, //, %	
+ (addition), - (subtraction)	lowest

• Augmented Assignment Operations

Combine operations together ⇒ machine do not have to call out a new address for x ⇒ Use same address

Operator	Expression	Identical Expression	English description
+=	x = 7 x += 2	x = 7 x = x + 2	x refers to 9
-=	x = 7 x -= 2	x = 7 x = x - 2	x refers to 5
*=	x = 7 x *= 2	x = 7 x = x * 2	x refers to 14
/=	x = 7 x /= 2	x = 7 x = x / 2	x refers to 3.5
//=	x = 7 x //= 2	x = 7 x = x // 2	x refers to 3
%=	x = 7 x %= 2	x = 7 x = x % 2	x refers to 1
**=	x = 7 x **= 2	x = 7 x = x ** 2	x refers to 49

NOTE x = 7 ; x **= 2 ⇒ variable x is now assigned 49

Programming Process

Friday, 14 January 2022 15:21

1. Readability Tips (# clean code)

+) Use white space to separate variables and operators
+) Be consistent
+) Pick variable names that are easy to read and interpret
+) Be consistent with naming schemes
+) Single Statement that spans multiple lines

In order to split up a statement into more than one line, you need to do one of two things:

1. Make sure line break occurs inside parentheses
2. Use line-continuation character, which is a backlash \ (not divide /)

ex

```
>>> room_temperature_c = 20
>>> cooking_temperature_f = 350
>>> oven_heating_rate_c = 20
>>> oven_heating_time = (
... ((cooking_temperature_f - 32) * 5 / 9) - room_temperature_c) / \
... oven_heating_rate_c
>>> oven_heating_time
7.833333333333333
```

2. Comments

Use (#)

3. Testing

+) Golden Rule: Never program more than 15 minutes without testing
+) Modular Code (Divide code into blocks - modules)

4. Error Reduction vs Debugging

+) Reduce number of errors
+) Identifying and correcting errors

Planning Code

- How do you start writing code?
  - Read the question carefully and with intent
  - Think about what information was provided in the topic that you should include in your answer
  - Brainstorm different ways to answer the question
  - Skim through course material to see what could help
  - Scaffold or quickly structure each paragraph
  - Figure out what you want to conclude and think of ways to get there
  - Make sure each section has purpose (you aren't repeating yourself)
  - Think about order (what needs to be said at the beginning vs what needs to be said at the end)

Types of Error

Friday, 14 January 2022 15:29

Syntax Errors

- *Syntax error*: results when the programming language cannot understand your code.
- Examples: missing an operator or two operators in a row, illegal character in a variable name, missing a parentheses or bracket etc.
- In English, a syntax error is like a spelling error

```
>>> 3) + 2 * 4
Syntax Error: unmatched ')':  line 1, pos 2
```

Semantic Errors

- *Semantic error*: results from improper use of the statements or variables.
- Examples: using an operator not intended for the variable type, calling a function with the wrong argument type, or wrong number of arguments, etc.
- In English, a semantic error is like a grammar error

```
>>> "Hello" - 4
TypeError: unsupported operand type(s) for -: 'str' and 'int'
>>> number = number * 2
NameError: name 'number' is not defined
```

→ Calling wrong number of arguments in function.

Runtime Errors

- *Runtime error:* is an error that occurs during the execution (runtime) of a program. Generally do not occur in simple programs.
- The code could run fine most of the time, but in certain circumstances the program may encounter an unexpected error and crash.
- Examples: infinite loops, attempting to access an index out of bounds, etc.

```
>>> x = 10
>>> while x > 0:
        print("This is the song that never ends")
```

- Divide by 0 ⇒ Runtime error
- str + int ⇒ Runtime error

Logical Errors

- *Logical Error:* results from unintended result due to a miscalculation or misunderstanding of specifications.
- Examples: miscalculation, typo, misunderstanding of requirements, indentation mistakes, operator precedence, integer instead of floating-point division, etc.
- **Most difficult to fix** because the code will execute without crashing. There are no error messages produced.

Logical Error Examples

71.6 degrees F is about 22 degrees C

```
>>> fahrenheit = 71.6
>>> celsius  = fahrenheit – 32 * 5/9
>>> celsius
53.822222222222216
```

Correct logic: celsius = (fahrenheit – 32) * 5/9

```
>>> fahrenheit = 716
>>> celsius  = (fahrenheit – 32) * 5/9
>>> Celsius
380.0
```

Whoops, typo! Forgot the decimal.