Python Introduction
Let's Dive In!

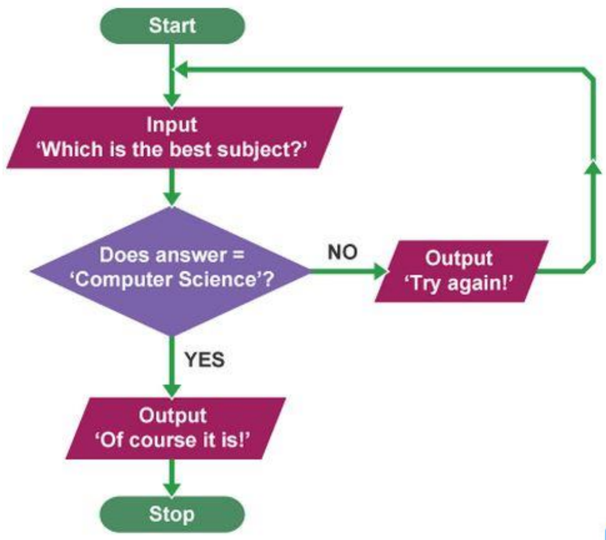
Remember computer programming is about **solving problems**. We can solve tons of cool problems using Python. You will eventually learn to create really neat tools and fun applications using Python.

But first we have to **learn some basics**. We need to get familiar with some key principles before we can use Python effectively. The next few lessons are designed to get you familiar with the basics of Python.

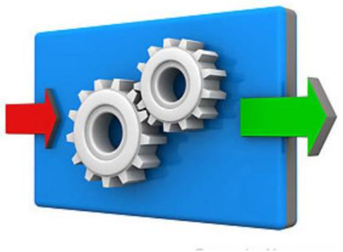
All computer programming languages, really, have a few key aspects:

1. **Input**
2. **Output**
3. **Decisions**
4. **Calculations**
5. **Repetition**

Python will give you tools in **all** of these areas. Let's take a look!

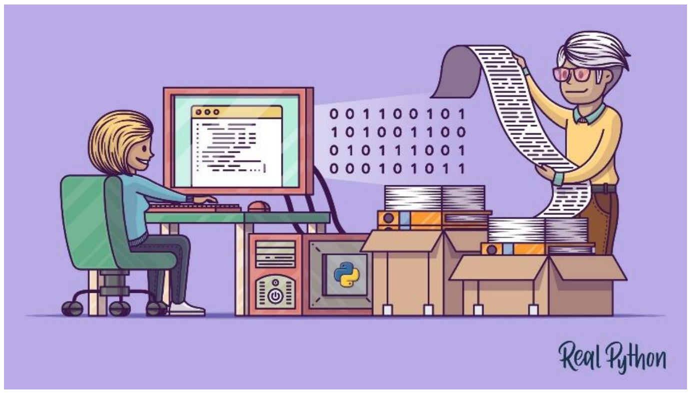


Input/Output/Variables:



Output:

Generally, Python can **output** (produce): lists, numbers, words, and images. The most interesting and easiest are outputs are **words** and **images** so let's start by practicing creating these simple **outputs**.



Exercise#1

Type the following into trinket (don't cut and paste! – Type it in please)

```
print('I am a cool student.')
print('I am learning Python.')
print('Python is fun!’)
```

Type the following into trinket (don't cut and paste! – Type it in please)

```
print('Im the best student in the world! ')
print('\n')
print('I am learning Python.')
```

make sure you know what '\n' does?

Type the following into trinket

```
print('Im the best \r student in the world! ')
print('I am learning Python. ')
```

Please **save** your exercises **in trinket** so you can submit all exercises as a single assignment later.

```
What does \r do?
\ and \n are called:
```

Escape Characters

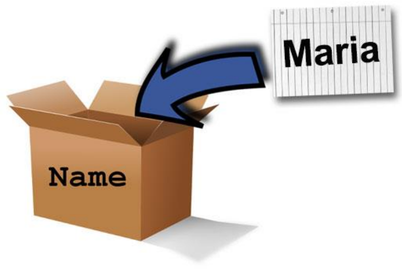
Escape Characters are special characters that help us organize written output. There are many of these. Here is a short list of some you.

Escape character	Prints as
`\'`	Single quote
`\"`	Double quote
`\t`	Tab
`\n`	Newline (line break)
`\\`	Backslash
`\r`	Return Carriage similar to Newline

Now you can get words to appear on a screen using Python. But that's not *too* exciting… you can do that by just typing into a word doc! Let's keep going:

Too Easy

Input/Variables:



One way for Python to gather information is by getting it from **humans**. Humans can type words, numbers, lists, and commands into python. We can then store these items in **variables.**

A **variable** (in all computer programming languages) is: a way to label and store information…kinda like putting something into a labelled box. Let's have a look to see how this works:

Exercise#2

Type the following into trinket

```
age=17
name='Jeff'
fav_food='apples'

print(age)
print(name)
print(fav_food)
                    #change my favorite food :)
fav_food='banana'
print(fav_food)

print("\n")         # leave a space
fav_food='toast'    # changed my favorite food again!
print('my name is')
print(name)
print('my favorite food is')
print(fav_food)
```

When you see a hashtag # in python, this means you have created a "comment" in your code. Comments are ignored by the computer, but are handy for humans who like to put comments and reminders in their code

See how we are **creating storage spaces** and then **stuffing things into them**? We do this with the = (equals sign)…which in programming does **NOT** mean "equal to"…it means *"assign value of"*

Type the following in to trinket!...more fun examples with **Variables**:

```
team_players=23
extras=5
total= team_players + extras
print('The number of regular players on our team is')
print(team_players)
print('We have this may extra players')
print(extras)
print ('Total amount of players is')
print(total)
print('\r')
extras=10
total= team_players + extras
print ('Now we have his many extas!')
print(extras)
print('So our total number of players is')
print(total)
```

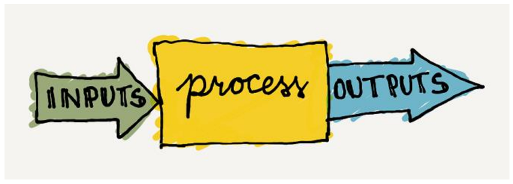
Now you can:

1. print stuff to the screen and
2. use **variables!**

Good for you!

Later we will look at how to **draw images** with python, but for now let's look at ***input***:

Input (Collecting information)

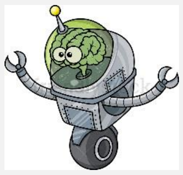


Collecting information is a key part of computer programming. One way Python does this is with the **`input()`**command.

The **`input()`** command allows python to **accept typed info from a human.**

Exercise#3

Enter the following code into trinket. When you run the program, the **`input()`** command will prompt you to enter information. Try it out!



```
print ('Hi, My name is Calcutron, Im from planet Zerp')
name=input('what is your name?')
print ('Hi',name)
planet=input('What is the name of your planet?')
print(planet,'?????')
print('Doesnt sound too cool. Im naming your planet Salad Ball!')
print('You live on Salad Ball!')
print('You like that',name,'?')
answer=input('yes or no?')
if answer=='yes':
  print('Good!',name,'Who is your King on planet Salad Ball?')
if answer=='no':
  print(name,'!!!! dont be such a poor sport!')
  print('Salad Ball!'*3)
```

Notice the "**if**" statements above…we will talk about later but it's not to hard to see what and '**if**" statement does from the example. ***INDENTING!*** – notice the ***indenting*** – very, very, very important!

Exercise#4 *(please save your work)*

- Create a new program similar to the one above (you can use the code above as a template). Your goal will be to **create a character that someone can have a dialog with.**
- Use `If` statements (don't forget about indenting).
- Make is as ***fun and as interesting as possible*** for the user!
- Show Mr. Walzl when you are done. You will be marked on this!
- Need a creative boost? Maybe use the scenario of a person going to buy something from the corner store.

Fun with Formatting:

Making stuff look good and easy to read on the screen is actually an important part of programming. Creating an environment where humans can easily interact with your software it crucial. We also want to be able to engage users and **clearly** communicate solutions to the problems our programs solve.

Below is a few ways to **help make your OUTPUT more readable and organized** from a user's point of view:

Pausing Your Program:

When creating cool stuff using python, you will sometimes want your program to **pause**. This can be useful when you want the program to wait a bit between outputs.

Exercise#5 (type the following into trinket and then answer the questions on the next page in the comment section below your code yourself)

```
import time
print('hello!')
time.sleep(1)
print('What\'s up?')
time.sleep(1)
answer=input('what’s your favorite color?')
time.sleep(1)
print('your favorite color is {}?'.format(answer))
#hashtag-(#) indicates comments that won’t be run as part of your program
#anything after the hashtag should only notes for the programmer.
#put your answers to the exercise questions on the next page in
#the notes below this code when you submit it.
```

What is happening in this example? Lots of stuff:

1. We **`imported`** the **`time`** "*library*". This is necessary to make the **`sleep`** function work.
2. We used the `time.sleep()` function to **pause** the program. The number `(1)` in the brackets is used to indicate the number of seconds the program will pause for.
3. We used the `{ }` and the `.format` function to insert a variable into a print statement....cool!
4. We used # (hashtags) to place comments in our code.

Exercise#5 **QUESTIONS:**

Write the answers to these questions as comments at the end of your exercise#5 python code in trinket that you using the # hashtag symbol
Make sure you **submit your code.**

1. What is a library in python (look it up)? Why do we have to import the time library for this code?
2. Change the (1) in the code to (2) what happens
3. Describe what the `{ } .format` command does.
4. What color are the comments in trinket when you use the # symbol?

Clearing the screen:

This function is particularly useful when you need to display several pieces of information on the screen at different times:

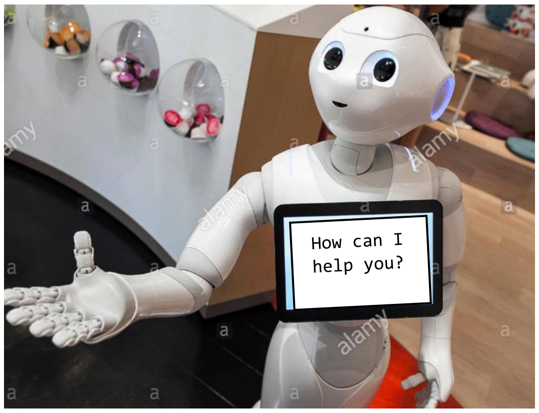
Exercise#6

Type the following into trinket (don't cut and paste! – Type it in please and SAVE)

```
import os
import time
print('Hello there!')
time.sleep(2)
os.system('clear')
name=input('What is your name?')
time.sleep(1)
os.system('clear')
print('Hi {}! Nice to see you today!'.format(name))
time.sleep(4)
os.system('clear')
print('Pleasant weather we\'re having isn\'t it?')
```

What's happening in the example is above?

- We have to `import os` (operating system library) and `time` library
- We used the `os.system('clear')` command to clear the screen
- We also used `time.sleep()` to pause the program at appropriate spots



Stand out!

Another cool formatting feature.

Want to really jazz up your **output** try the following:

Exercise#7

Type the following into Trinket to see what it does.
Save your work. Watch your **indenting!**

```
import time
import sys

sentence=('Check this out! What a cool way to format output in
Python!')
for i in sentence:
  sys.stdout.write(i)
  time.sleep(0.03)
```

What happening in the example is above?

- We have to `import sys` (system library) and the `time` library
- We use a "`for` loop" to repeat some instructions…more about this later.
- We used the `sys.stdout.write(i)` so we can keep write each character out on the same line
- We also used `time.sleep(0.03)` to pause the program for 0.03 seconds after each character is written to the screen at appropriate spots

Exercise#8

Create a cool dialog between a robot and a human ***similar to what you did in exercise#4***, but this time make sure you include the following:

- Use the **clear screen** and **sleep functions** in your program (appropriately).
- Use `if` statements
- Use variables
- `Use the { }` and the `.format()` commands to insert variables into text.
- Use the `stdout` function.
- Make sure the dialog is as engaging and fun as possible.